AD-A107 858 TEXAS TECH UNIV LUBBOCK DEPT OF MATHEMATICS F/G 12/1
LIE GROUPS AND LIE ALGEBRAS IN VIDEO TRACKING.(U)
OCT 81 T G NEWMAN DAAG29-80-C-0087
UNCLASSIFIED ARO-17219.1-R-EL NL

1 OF 1
AD A
107858

END
DATE
FILMED
01-82
DTIC

ARO 17219.1-R-EL

LEVEL (12)

AD A107858

SEAL OF TEXAS TECH UNIVERSITY

81 11 24 073

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Lie Groups and Lie Algebras in Video Tracking		5. TYPE OF REPORT & PERIOD COVERED Final 3/1/80 - 2/28/81
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Thomas G. Newman		8. CONTRACT OR GRANT NUMBER(s) DAAG 29-80-C-0087
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Mathematics, Texas Tech University, Lubbock, TX 79409		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709		12. REPORT DATE 10-26-81
		13. NUMBER OF PAGES 58
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release;
Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official department of the Army position, policy, or decision unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Video tracking, two dimensional image, Lie group, Lie algebra, differential form, affine transformations

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

When a three-dimensional dynamic scene is projected to a two-dimensional image plane, a complex class of motions are observed, which may be approximated by translation and linear transformation. In this report, we show how such motion may be resolved into principal components and measured from 2-dimensional data. Lie theoretic techniques are used to obtain a motion model, and to incorporate generalized velocity measurements in a closed loop feedback tracker. Stability of numerical calculations is enhanced by a method based on an application of Stoke's Theorem to certain differential 2-forms. New 412066

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Lie Groups and Lie Algebras in Video Tracking

by

Thomas G. Newman

October 26, 1981

U.S. Army Research Office

Contract DAAG29-80-C-0087

Department of Mathematics

Texas Tech University

Lubbock, Texas 79409

Table of Contents

I. Affine Transformations and Tracking.

When a dynamic three-dimensional scene is observed via an optical projection, a quite complex class of motions are induced in the image plane [5,6,7,9]. In general, this class of motions is highly non-linear, being dependent on the geometry of the objects being observed as well as their trajectories in space[3]. Nevertheless, in many cases the motion is approximated closely by translation, magnification and rotation in the image plane. It is easy to see that this approximation is best for motion in space which consists of translation and rotation about a line parallel to the bare sight.

A better approximation results by consideration of the full affine group in the plane, which includes shearing in two directions as well as the motions mentioned above. By definition, an affine transformation in the plane R^2 is of the form

$$T(y) = Ay + a, \ y \varepsilon R^2 \tag{1.1}$$

where A is a non-singular 2x2 matrix and $a \varepsilon R^2$ is considered as a column vector [4,10]. The set of all such transformations T is called the <u>general affine group</u> and is denoted GA(2). It is easily seen that the subset consisting of translations, magnifications and rotations forms a subgroup, which we denote by SA(2). In order that $T(y) = Ay + a$ belong to SA(2) it is necessary and sufficient that $A_{11} = A_{22}$ and $A_{12} = -A_{21}$. In this case, the magnification factor is $(A_{11}^2 + A_{21}^2)^{1/2}$ and the rotation angle is $\mathrm{atn}(A_{21}/A_{11})$.

In order to consider dynamic images, it is necessary to allow

A and a in (1.1) to depend on time. This gives rise to a trajectory $u(t,y)$ for each $y \varepsilon R^2$ given by

$$u(t,y) = A(t)y + a(t) \tag{1.2}$$

where $(A(t),\ a(t)) \ \varepsilon\ GA(2)$, and we require that $A(0)=I$, $a(0)=0$ in order that the trajectory pass through y at time $t=0$; i.e., $u(0,y) = y$.

As in [4], though only for linear transformations, we may realize the pair $(A(t),\ a(t))$ as the solution of a linear system of differential equations. Let us define

$$\Lambda(t) = \dot{A}(t)\ A^{-1}(t) \tag{1.3a}$$

$$\lambda(t) = \dot{a}(t) - \Lambda(t)\ a(t), \tag{1.3b}$$

from which,

$$\dot{A}(t) = \Lambda(t)\ A(t),\ A(0) = I \tag{1.4a}$$

$$\dot{a}(t) = \lambda(t) + \Lambda(t)\ a(t),\ a(0) = 0 \tag{1.4b}$$

We may summarize the correspondences defined by (1.3) and (1.4) as follows:

Theorem 1.1: Equations (1.3) and (1.4) establish a one-to-one correspondence between differentiable curves $(A(t),\ a(t))$ in $GA(2)$ satisfying $A(0) = I$, $a(0) = 0$ and continuous curves $(\Lambda(t),\ \lambda(t))$ where $\Lambda(t)$ is an arbitrary 2x2 matrix and $\lambda(t)\ \varepsilon R^2$. Moreover, in order that (A,a) belong to $SA(2)$ it is necessary and sufficient that $\Lambda_{11} = \Lambda_{22}$ and $\Lambda_{12} = -\Lambda_{21}$.

The first part of the above theorem apparent from (1.3) and (1.4). A rigorous and detailed proof proceeds exactly as given in [4] for linear transformations. The last part can be deduced by a few calculations using the fact that elements of $SA(2)$ satisfy $A_{11} = A_{22}$ and $A_{12} = -A_{21}$.

Now, if we differentiate (1.2) with respect to t, use (1.4) and (1.2) again, we obtain

$$\frac{\partial u}{\partial t}(t,y) = \Lambda(t)\ u(t,y) + \lambda(t). \tag{1.5}$$

Of course, $v(t,y) = \frac{\partial u}{\partial t}(t,y)$ is the velocity field along the trajectory $u(t,y)$. Equ. (1.5) shows that the velocity at a point u depends on u as well as t and is therefore not spatially invariant.

Note that (1.5) in fact gives the differential equation for an arbitrary affine trajectory, and when $\Lambda(t)$ is restricted as in Theorem 1.1, it gives the equation for a trajectory under the restricted group of motions SA(2). By virtue of (1.2), we see that $(A(t), a(t))$ obtained from (1.4) may be considered as a fundamental system of solutions to the evolution equation (1.5). Now it is important to note that the fundamental system of solutions is completely determined by the pair $(\Lambda(t), \lambda(t))$ which is spatially invariant, being a function of time only. To establish convenient terminology, let us give the following

<u>Definition 1.1</u>: The pair $(\Lambda(t), \lambda(t))$ is the <u>generalized velocity field</u> of the family of affine trajectories $u(t,y)$ defined by (1.5).

We may now state

<u>Theorem 1.2</u>: A family $u(t,y)$ of affine trajectories satisfying $u(0,y) = y$ is completely determined from its generalized velocity field, which is spatially invariant, via (1.4) and (1.2). Moreover, the absolute velocity v at a point u on a trajectory is given, as in (1.5), by $v = \Lambda(t)u + \lambda(t)$.

Let us write $\lambda(t) = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$, $\Lambda(t) = \begin{bmatrix} \lambda_3 & \lambda_5 \\ \lambda_4 & \lambda_6 \end{bmatrix}$ and expand the equation $\dot{u} = \Lambda u + \lambda$ (where the t-dependence has been suppressed) in the form

$$\frac{\partial}{\partial t}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \lambda_1\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \lambda_2\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \lambda_3\begin{bmatrix} u_1 \\ 0 \end{bmatrix} + \lambda_4\begin{bmatrix} 0 \\ u_1 \end{bmatrix} + \lambda_5\begin{bmatrix} u_2 \\ 0 \end{bmatrix} + \lambda_6\begin{bmatrix} 0 \\ u_2 \end{bmatrix} \tag{1.6}$$

In this way we can identify individual vector fields $v^1(u), \ldots,$ $v^6(u)$ and write (1.6) in the form

$$\frac{\partial u}{\partial t} = \sum_{i=1}^{6} \lambda_i(t)\ V^i(u) \tag{1.7}$$

In a similar manner for SA(2), we write

$$\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} \text{ and } \Lambda = \begin{bmatrix} \lambda_3 & -\lambda_4 \\ \lambda_4 & \lambda_3 \end{bmatrix}$$

so that

$$\frac{\partial}{\partial t}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \lambda_1\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \lambda_2\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \lambda_3\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \lambda_4\begin{bmatrix} -u_2 \\ u_1 \end{bmatrix}, \tag{1.8}$$

allowing four vector fields $v^1(u), \ldots, v^4(u)$ to be indentified. Finally, we rewrite (1.8) as

$$\frac{\partial u}{\partial t} = \sum_{i=1}^{4} \lambda_i(t)\ v^i(u) \tag{1.9}$$

It should be noted that the functions v^i defined by (1.7) or (1.9) are characteristic of the class of motions under consideration, and more general classes of motion can be treated by consideration of other generators $v^1, v^2, \cdots$. In the cases of interest, the sets of vector fields derived above define the Lie algebras [2] of GA(2) and SA(2). Any vector field $v: R^2 \to R^2$ induces a differential operator Y_v, called an infinitesimal transformation, which is defined by

$$Y_v = v_1(y)\frac{\partial}{\partial y_1} + v_2(y)\frac{\partial}{\partial y_2} \tag{1.10}$$

where $v_1(y)$ and $v_2(y)$ are the components of $v(y)$. In Tables 1 and 2 we list the infinitesimal transformations for the groups GA(2) and SA(2), given in terms of a variable $x=(x_1,x_2)$ for later application

Table 1. Infinitesimal transformations for GA(2).

$X_1 = \frac{\partial}{\partial x_1}$	$X_3 = x_1 \frac{\partial}{\partial x_1}$	$X_5 = x_2 \frac{\partial}{\partial x_1}$
$X_2 = \frac{\partial}{\partial x_2}$	$X_4 = x_1 \frac{\partial}{\partial x_2}$	$X_6 = x_2 \frac{\partial}{\partial x_2}$

Table 2. Infinitesimal transformations for SA(2)

$X_1 = \frac{\partial}{\partial x_1}$	$X_3 = x_1 \frac{\partial}{\partial x_1} + x_2 \frac{\partial}{\partial x_2}$
$X_2 = \frac{\partial}{\partial x_2}$	$X_4 = x_1 \frac{\partial}{\partial x_2} - x_2 \frac{\partial}{\partial x_1}$

Note that $u(t,y)$ given by (1.2) may be regarded as the location at time t of the particle which was at y at time t=0. An observar at some point x will observe this particle provided that $x = u(t,y) = A(t)y + A(t)$. We may solve this equation for y to obtain $y = A^{-1}(t)(x - a(t))$. Thus we define the trace of the point $x \in R^2$ to be

$$s(t,x) = A^{-1}(t)(x-a(t)),\ x\varepsilon R^2. \tag{1.11}$$

We may interpret $s(t,x)$ to be the particle which will arrive at x at time t.

Let us now consider a two-dimensional image, represented by a function $f:R^2 \to R$, and suppose that the image f is subjected to an affine transformation $(A(t),\ a(t))$. Here a value $f(y)$ is regarded as a feature which propogates along the trajectories of the motion. This is an extremely powerful, and somewhat restrictive, assumption which is not always valid in real images. For example, it is violated by changes in radiance values which vary as a function of the angle of incident illumination. On the other hand, it is valid in most instances over short time intervals and deviations from this assumption may frequently be treated as higher order effects.

In any event, if the feature $f(y)$ is propagated along trajectories, then a stationary observor, say at point x, will observe a value $F(t,x) = f(s(t,x))$ at time t, since $s(t,x)$ represents the particle arriving at x at time t. We may now state a most important result.

Theorem 1.3: Let a time-varying image F be given by

$$F(t,x) = f(s(t,x)) \tag{1.12}$$

where $s(t,x)$ is an affine trace as in (1.11) with generalized velocities

$$\lambda(t) = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}, \Lambda(t) = \begin{bmatrix} \lambda_3 & \lambda_5 \\ \lambda_4 & \lambda_6 \end{bmatrix}. \text{ Then}$$

$$-\frac{\partial F}{\partial t} = \sum_{i=1}^{6} \lambda_i(t)\, X_i\, F\,, \tag{1.13}$$

where $X_1,\ldots,X_6$ are given in Table 1.

A similar result holds if we restrict to SA(2), using $X_1,\ldots,X_4$ from Table 2.

Proof: This result may be deduced from results given in [7], provided we compensate for the change from "left invariance" in that development to the "right invariance" of the current treatment. However, a direct proof is instructional and will be outlined herein. We first show

Lemma 1.3.1: For $s(t,x)$ given by (1.11)

$$-\frac{\partial s}{\partial t} = \sum_{i=1}^{6} \lambda_i(t) X_i s. \tag{1.14}$$

Proof: First note that by direct calculation we have $\sum\lambda_i X_i s = \sum\lambda_i X_i A^{-1}(x-a) = A^{-1}(\sum\lambda_i X_i x) = A^{-1}(\sum\lambda_i v^i(x)) = A^{-1}(\Lambda x+\lambda)$. Also, noting that $\frac{d}{dt} A^{-1} = -A^{-1}\dot{A}A^{-1}$, we have $\frac{\partial s}{\partial t} = \frac{\partial}{\partial t} A^{-1}(x-a) = -A^{-1}\dot{A}A^{-1}(x-a)-A^{-1}\dot{a} = -A^{-1}\Lambda(x-a) - A^{-1}(\lambda + \Lambda a) = -A^{-1}(\Lambda x + \lambda)$. Hence, the desired result follows.

Returning to the proof of Theorem 1.3, we have

$$\sum_i \lambda_i X_i F(tx) = \sum_i \lambda_i(t) v_j^i(x) \frac{\partial}{\partial x_j} f(s(t,x))$$

$$= \sum_i \lambda_i(t)\; v_j^i(x)\; \frac{\partial s_k}{\partial x_j}(t,x)\; \frac{\partial f}{\partial s_R}(s)$$

$$= \sum_i \lambda_i(t)\ X_i s_R(t,x)\ \frac{\partial f}{\partial s_k}\ (s)$$

$$= - \frac{\partial s_k}{\partial t}\ (t,x)\ \frac{\partial f}{\partial s_k}\ (s)\ = - \frac{\partial f(s(t,x))}{2t}$$

$$= - \frac{\partial F}{\partial t}\ (t,x)\ \text{, as desired.}$$

Theorem 1.3 appears to be fundamental to the analysis of motion in dynamic images. As is evident from the proof, an analogue is valid in a much more general setting. In fact, scrutiny of the proof shows that it depends mainly on Lemma 1.3.1. Consequently, the theorem will hold for any class of motions for which a suitable form of the "trace" lemma can be obtained. The significance of Theorem 1.3 lies chiefly in the fact that the generalized velocities (which are usually unknown) appear as linear coefficients in (1.13), along with quantities which can be calculated from the data $F(t,x)$.

The main problem with the extraction of the generalized velocities from (1.13) is the general lack of numerical precision in the calculation of the derivatives from real data (e.g., digitized video). In subsequent sections we shall show how to incorporate (1.13) in a feedback loop which is very stable and how to obtain an equivalent formulation based on integration rather than differentiation.

II. A Velocity Feedback Tracker

The theoretical results of this section result from research done under a separate contract* and for which a publication is in preparation. In view of the fact that the techniques have been incorporated in the experimental portion of this report, these results will be presented in this report in the context of affine transformations.

Let absolute image coordinates be denoted by $y = (y_1, y_2)$ and introduce additional coordinates as follow: Let coordinates z be established relative to a moving target, and let coordinates x be established in a movable "window". We assume that the motions of both the target and the window may be described by affine transformations relative to absolute image coordinates. It is assumed that the motion of the window may be chosen at will, while the motion of the target is prescribed (e.g. by nature) and is unknown.

By the affine assumption, we may describe the transformation from window coordinates to image coordinates by

$$y_W(t,x) = A(t)x + a(t) \quad , \; x \varepsilon R_W^2 \; , \tag{2.1}$$

where (A(t), a(t)) is a suitable family of affine transformations. Similarly, the transformation from target coordinates to image coordinates is

$$y_T(t,z) = B(t)\, z + b(t) \quad , \; z \varepsilon R_T^2 \tag{2.2}$$

for suitable (B(t), b(t)) ε GA(2). Let us denote the respective generalized velocity fields by (Λ_A, λ_A) and (Λ_B, λ_B).

By equating $y_T(t,z) = y_W(t,x)$ we may solve for the point z

* ONR Contract N0014-76-C-1136

on the target which arrives at point x in the window at time t, to obtain:

$$z(t,x) = B^{-1}(Ax + a - b), \tag{2.3}$$

where dependence on t has been suppressed on the right. We note that z(t,x) in (2.3) may be regarded as a trace in the sense of the previous section. By an application of Lemma 2.3.1 we have:

Theorem 2.1: There exists a generalized velocity field $(\Gamma(t),\gamma(t))$ such that

$$-\frac{\partial z(t,x)}{\partial t} = \sum_{i=1}^{6} \gamma_i(t)\; X_i z(t,x), \tag{2.4}$$

where $\gamma(t) = \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}$, $\Gamma = \begin{bmatrix} \gamma_3 & \gamma_5 \\ \gamma_4 & \gamma_6 \end{bmatrix}$ and the operators $X_1,\ldots,X_6$ are given in window coordinates as in Table 1.

Now, let f(z) be a feature of the target, measured at point z, and assume that this feature propogates along the target trojectories. An observor at point x in the window therefore observes data F(t,x) = f(z(t,x)), inasmuch as z(t,x) is the point which arrives at x at time t. From Theorem 1.3 we obtain

Theorem 2.2: In the above context,

$$-\frac{\partial F}{\partial t}(t,x) = \sum_{i=1}^{6} \gamma_i(t) X_i F(t,x) \tag{2.5}$$

In principle, (2.5) allows the determination of the generalized velocities of the target relative to the window. Since we have free choice of window velocities, relative to the image coordinates, this is tantamount to measurement of absolute target velocities. The conversion process will now be described.

Let us denote by $\bar{\underline{X}}$ the window space, $\bar{\underline{Y}}$ the image space, and let $T(\bar{\underline{X}})$ and $T(\bar{\underline{Y}})$ be the respective tangent spaces. Since the map from $\bar{\underline{X}}$ to $\bar{\underline{Y}}$ is $y = A\,x + a$, as in (2.1), it follows that the induced map on the tangent spaces is simply $y^* = A\,x^*$ [1,2]. Now the velocity field (Γ, γ) defines a map $\bar{\underline{X}} \to T(\bar{\underline{X}})$ given by $x^* = \Gamma\,x + \gamma$ (see (1.5)). Accordingly, a velocity field (∇, λ) is induced on $\bar{\underline{Y}}$, which maps $\bar{\underline{Y}} \to T\,(\bar{\underline{Y}})$ in a similar fashion. The velocity field (Λ, λ) is defined by the commutative diagram

$$\begin{tikzcd}
\bar{\underline{X}} \arrow[r, "(\Gamma,\gamma)"] \arrow[d, "(A,a)"'] & T(\bar{\underline{X}}) \arrow[d, "A"] \\
\bar{\underline{Y}} \arrow[r, "(\Lambda,\lambda)"] & T(\bar{\underline{Y}}).
\end{tikzcd}$$

Thus, we calculate $y^* = \Lambda\,y + \lambda$, by inverting (A,a) and taking the upper path, to be given by $y^* = A(\Gamma\,A^{-1}(y-a)+\gamma) = A\Gamma A^{-1}y + A\gamma - A\Gamma A^{-1}a$. By comparison, we obtain

$$\Lambda = A\Gamma A^{-1} \tag{2.6a}$$

$$\lambda = A\gamma - A\Gamma A^{-1}a \tag{2.6b}$$

Now, since the velocity field (Γ,γ) represents the difference between target and window velocities in the window coordinate system, we see that (Λ,λ) must represent this same difference relative to the absolute image coordinate system. That is,

$$\Lambda = \Lambda_B - \Lambda_A \tag{2.7a}$$

$$\lambda = \lambda_B - \lambda_A \tag{2.7b}$$

We may summarize these results in a useable form as follows:

Theorem 2.3: Let (2.1) and (2.2) define the motion of a window and a target, respectively, relative to a system of absolute image coordinates, and let (Λ_A, λ_A) and (Λ_B, λ_B) be the corresponding generalized velocities. Further, let (Γ, γ) be the generalized velocities of the target relative to the window, as determined by (2.5).

Then

$$\Lambda_B - \Lambda_A = A\Gamma A^{-1} \tag{2.8a}$$

$$\lambda_B - \lambda_A = A\gamma - A\Gamma A^{-1}a. \tag{2.8b}$$

The previous theorem immediately suggests an algorithm for determination of velocities in an image. More generally, the algorithm performs tracking since, as will be seen, the result is to force the window to follow the target by emulation of velocities. The algorithm is as follows:

Step 1. Initialize the window by choice of $A(0)$, $a(0)$. In the absence of a priori information, initialize $\Lambda_A(0)=0$, $\lambda_A(0)=0$. Sample window values $F(t_0,x)$ at time $t_0=0$.

Step 2. Sample window values $F(t_n,x)$ at time $t_n=t_{n-1}+\delta$. Approximate $\frac{\partial F}{\partial t}$ and $X_i F$ at various points in the window and form a system of linear equations using (2.5).

Step 3. Solve the resulting linear equations for $\gamma_1;\gamma_2,\dots$.

Step 4. Replace $\Lambda_A \leftarrow \Lambda_A + A\Gamma A^{-1}$ and $\lambda_A \leftarrow \lambda_A + A\gamma - A\Gamma A^{-1}a$.
Note: If the calculation of $\gamma_1, \gamma_2 \dots$ were exact, this would result in $\Lambda_A \leftarrow \Lambda_B(t_n)$ and $\lambda_A \leftarrow \lambda_B(t_n)$

Step 5. Take a δ time step in the numerical solution $\dot{A} = \Lambda_A A$, $\dot{a} = \gamma_A + \Lambda_A a$ to obtain $(A(t_n), a(t_n))$. This effectively moves the window.

<u>Step 6</u>. Repeat from step 2.

Emperical results indicate that the above algorithm conveys rapidly over a fairly broad range of target velocities. Although the initial estimate of target velocities is usually fairly coarse, it is generally in the right direction and results in good estimates after 3 to 5 iterations. Subsequently, the target is tracked very well with only a nominal amount of slew. More importantly, the computational speed is such that it is feasible for real-time implementation, with calculations having been done at 25 to 100 iterations per second on various computers, including time spent in simulation support.

The most notable failure is a high degree of instability encountered in dealing with real data in the form of digitized images. The available image data, however, did not have a suitable dynamic range in comparison to the noise level. Considerable improvement resulted by expanding the contract and filtering to obtain a greater dynamic range.

The results of performing the above algorithm on simulated data is presented in appendix A.

III. Alternate Formulation via 2 - forms.

The major source of error in the calculation of generalized velocities would appear to be that introduced in the numerical approximation of spatial derivatives. Although the situation is improved somewhat by filtering and the use of multi-point formulas, it is still desirable to seek alternate approaches. As can be seen from examination of the algorithm of the preceeding section, any method for calculation of generalized velocities may easily be inserted in the basic tracker.

In this section we appeal to a form of Stoke's Theorem [1] to obtain an integration based analogue of Theorem 1.3. The formula obtained is strictly valid only when the generalized velocities are constant, although is is a useful approximation when the rates of change of the velocities are small.

We consider the three dimensional space R^3 consisting of time t and two spatial variables x and y. Coordinates $\xi=(t,x,y)$ are chosen to make a right-hand coordinate system, and we observe this orientation in defining differential forms. We state the form of Stoke's Theorem required:

Stoke's Theorem: Let Ω be a rectangle in (t,x,y) space R^3 and let w be a differentiable 2-form. Then

$$\int_{\partial\Omega}\omega = \int_{\Omega} d\omega \tag{3.1}$$

Here ω is of the form $\omega = \alpha_0 dxdy + \alpha, dydt + \alpha_2 dtdx$, with α_0, α_1, α_2, differentiable functions of $\xi \varepsilon R^3$, and

$$d\,\omega = \frac{\partial\alpha_0}{\partial t} + \frac{\partial\alpha_1}{\partial x} + \frac{\partial\alpha_2}{\partial y} \quad dtdxdy.$$

Although the results to be presented may be generalized considerably, our derivation and experimental results will be given only for SA(2). Thus, the appropriate vector fields and corresponding infinitesimal transformations may be obtained from (1.8), (1.10) and Table 2, with x and y substituted in the obvious manner. By analogy with (1.9), for a constant velocity field $\lambda = (\lambda_1, \lambda_2, \lambda_3, \lambda_4)$, let us define a vector valued map $\eta: R^4 x R^2 \to R^2$ by

$$\eta(\lambda,\xi) = \sum_{i=1}^{4} \lambda_i v^i(\xi) = \begin{bmatrix} \lambda_1 + \lambda_3 x - \lambda_4 y \\ \lambda_2 + \lambda_4 x + \lambda_3 y \end{bmatrix} \quad . \tag{3.2}$$

As usual, let η_1, η_2 denote the components of η .

Now, if $s(t,\xi)$ is the trace corresponding to the generalized velocity field λ and $F(t,\xi) = f(s(t,\xi))$ is observed data, we may express Equ. (1.13) of Theorem 1.3 as

$$-\frac{\partial F}{\partial t} = \eta_1 \frac{\partial F}{\partial x} + \eta_2 \frac{\partial F}{\partial y} \ . \tag{3.3}$$

We intend to apply <u>Stoke's Theorem</u> to the 2-form defined by

$$\omega = F\,dxdy + \eta_1 F\,dydt + \eta_2 F\,dtdx \tag{3.4}$$

The principal result is stated as

<u>Theorem 3.1</u>: In the context above,

$$\begin{aligned} d\omega &= \left(\frac{\partial \eta_1}{\partial x} + \frac{\partial \eta_2}{\partial y}\right) F\,dtdxdy \\ &= 2\lambda_3 F\,dtdxdy \end{aligned} \tag{3.5}$$

To establish this, we calculate $d\omega$, using the fact that $dxdydt = dydtdx = dtdydx$ (whereas, for example, observing orientation, $dtdydt = -dtdxdy$). We have

$$d\omega = \left(\frac{\partial F}{\partial t} + \pi_1 \frac{\partial F}{\partial x} + F \frac{\partial \pi_1}{\partial x} + \pi_2 \frac{\partial F}{\partial y} + F \frac{\partial \pi_2}{\partial y}\right) dtdxdy.$$ By application of (3.3) and then (3.2) this simplifies to $dw = \left(F \frac{\partial \pi_1}{\partial x} + F \frac{\partial \pi_2}{\partial y}\right) dtdxdy = 2\lambda_3 F\, dtdxdy$, as desired.

By an application of Stoke's Theorem, and a somewhat tedious calculation, we immediately obtain:

Theorem (3.2). Let Ω be a rectangle in R^3 defined by opposing corners (t_1, x_1, y_2). In the context described above, in particular with λ constant, we have

$$\sum_{i=1}^{4} \lambda_i k_i = k_0 \tag{3.6}$$

where $k_0, k_1, \ldots, k_4$ are given in Table 3.

Table 3. Coefficients resulting from Stoke's Theorem.

$$k_0 = -\int_{\partial\Omega} F\, dxdy$$

$$k_1 = \int_{\partial\Omega} F\, dydt$$

$$k_2 = \int_{\partial\Omega} F\, dtdx$$

$$k_3 = \int_{\partial\Omega} xF\, dydt + \int_{\partial\Omega} yF\, dtdx - 2\int_{\Omega} F\, dtdxdy$$

$$k_4 = \int_{\partial\Omega} xF\, dtdx - \int_{\partial\Omega} yF\, dydt$$

It is important that orientation be considered in the evaluation of the coefficients in Table 3 (see [1]). The sign convention is such that for a principal 2-form (e.g., dxdy) a positive (negative) sign prevails on a face of the rectangle Ω provided that application of a right-hand rule points outward from (inward to) the rectangle Ω. Writing $\int_x$ for $\int_{x_1}^{x_2}$ (similarly for t and y) and assuming that $t_1 < t_2$, $x_1 < x_2$, $y_1 < y_2$, by way of example we have,

$$\int_{\partial\Omega} F\, dxdy = \int_y \int_x F(t_2,x,y)\, dxdy - \int_y \int_x F(t_1,x,y) dxdy,$$

and

$$\int_{\partial\Omega} xF\, dydt = x_2 \int_t \int_y F(t,x_2,y) dydt - x_1 \int_t \int_y F(t,x_1,y) dydt$$

and

$$\int_{\partial\Omega} xF dtdx = \int_x \int_t xF(t,x,y_2) dtdx - \int_x \int_t xF(t,x,y,) dtdx.$$

The remaining integrals may be expanded in a similar fashion.

Observe that differences are not entirely eliminated from the final formulas. However, the formulas are so written to indicate that the differences are taken after integration, even though in certain cases the formula could be collapsed with a difference taken before evaluation of the iterated integral.

The advantage of (3.6) over (1.13) as a means of calculation of the generalized velocities is achieved mainly by the filtering effect of the surface and volume integrals. As a matter of practice, several rectangles $\Omega_1, \ldots, \Omega_m$ are selected. Each rectangle Ω_e gives rise to an equation of the form (3.6),

$$\sum_{i=1}^{4} k_i^{(e)} \lambda_i = k_0^{(e)} \quad . \tag{3.7}$$

The resulting system of m equations in 4 unknowns may then be solved by a least-squares method. Note that this approach may be applied to the feedback tracking algorithm presented in Section 2 to calculate the velocities $\gamma_1, \ldots, \gamma_4$ of the target relative to the window, replacing the corresponding calculations based on (2.5). This has been implemented in a computer program and tested on real image data. The results are very encouraging and are presented in part in Appendix B. This method involves more computational overhead, with the best rate achieved to this point being about 10 iterations (=frames) per second. With some streamlining we believe that real-time rates of 30 frames per second can be achieved.

IV. Summary and Conclusions.

This report presents a method based on the theory of Lie groups for velocity tracking in a dynamic image in which the motion of picture parts can be ascribed to affine transformations. A feedback tracking algorithm was developed and tested on simulated data.

Since the random disturbances in real images preclude the use of simple methods for obtaining equations involving the velocities of trajectory, a method based on integration of differential forms was developed. This method was incorporated in the feedback tracker and tested on real image data. The results are very encouraging, with computation speeds approaching ten frames per second on a VAX 11/780. We believe that this method is viable as a component of a real-time video tracking system.

Among the problems left outstanding, a satisfactory algorithm for target acquisition has not yet been developed. In the experimental work performed, the initial target location was supplied as an input parameter. To be useful, a method for automating this step is essential.

In addition, we continue to experience problems with numerical percision. This seems to be related to the absence of sufficient dynamic range in real image data, indicating that this could be improved by changes in the capability of sensors. We feel that a great improvement would result from greater contrast in the image data.

Finally, the class of motions considered herein (affine or restricted affine) is not general enough for many applications, and the methods need to be extended to include projective distortion as well.

The equations which relate generalized velocities to time-varying images have other applications. They have been applied to a problem in pattern matching with considerable success, as fully described in [1]. The theoretical results and a summary of the experimental results of [11] have been submitted for publication as reference [8], which is attached as Appendix C.

References

[1] Buck, C.R. and E.F. Buck, Advanced Calculus, McGraw Hill Brook Co., New York, 1965.

[2] Cohn, P.M., Lie Groups, Cambridge University Press, London, 1957.

[3] Fishback, W.T., Projective and Euclidean Geometry, John Wiley and Sons, Inc., 1969.

[4] Greub, W.H., Linear Algebra, Springer-Verlag, Inc., New York, 1967.

[5] Huang, T.S., "Noise filtering in moving images", Workshop on imaging trackers and autonomons acquisition applications for missile guidance, Redstone Arsenal, Huntsville, AL, 1979.

[6] Martin, W., and J.K., Aggarwal, "Survey, dynamic scene analysis", Computer Graphics and Image Processing", Vol. 7, 1978.

[7] Newman, T.G., and D.A. Demus, "Lie theoretic methods in video tracking", Workshop on imaging trackers and autonomous acquisition applications for missile guidance", Redstone Arsenal, Huntsville, AL, 1979

[8] Newman, T.G. and L. Zlobec, "Adaptive pattern matching using control theory on Lie groups", Proceedings of the International Symposium on the Mathematical Theory of Networks and Systems, Santa Monica, CA, Aug. 1981.

[9] Roach, J.W., and J.K. Aggarwal, "Computer tracking of objects moving in space", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 1, 1979.

[10] Smirnov, V.I. (rev.,ed. R.A. Silverman), Linear Algebra and Group Theory, McGraw-Hill Book Co.,Inc., New York, 1961.

[11] Zlobec, Leopold, Pattern Matching by Means of Adaptive Control, Master's Report, Texas Tech University, 1980.

APPENDIX A

Feedback Tracker Simulation

A typical imaging system might include a sensor with a diameter (or cross section) of 25 mm and an optical focal length of 200 mm. At a 500 x 500 pixel density we obtain a conversion factor of .05 mm/pix or 20 pix/mm. With a target range of 1 km, say, then we obtain a conversion rate from sensor to target of 5 m/mm @ 1 km.

In the results to be presented, translation velocities may be regarded as being given in mm/sec. Conversion to pix/sec or m/sec at the target may be done by multiplication by the appropriate factor. Thus, a translation velocity of 7 mm/sec at the sensor corresponds to 20 pix/sec or to 5 m/sec at a target having a range of 1 km. A magnification velocity of 1/sec translates by the same factor and would therefore represent a velocity of 5m/sec @ 1 km toward the sensor. On the other hand, rotational velocity may be considered as given in radius/sec.

In Table A-1 we present the output of the tracking simulator (the output routines were modified slightly for ease of presentation). Note that the target velocities (10, -10, 10, 10) correspond to a 3-D object with a translational velocity of 86.6 m/sec @ 1 km (about 194 miles/hour) which is rotating about 3 revolutions per second about bore sight. The time base was chosen as 100 frames/sec. Inspection of the last four columns of Table A-1 shows that the target velocities have been acquired satisfactorily after only 3 frames, at t=.03, and subsequently refined to exact values.

Actual Target Velocities			
Hor Trans	Vert Trans	Magnification	Rotation
0.1000E+02	-0.1000E+02	0.1000E+02	0.1000E+02

	Horizontal	Vertical	Generalized Velocities			
Time	Position	Position	Hor Trans	Vert Trans	Magnification	Rotation
0.00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
0.01	0.8356E-01	-0.1050E+00	0.8356E+01	-0.1050E+02	0.7870E+01	0.5660E+01
0.02	0.1967E+00	-0.2069E+00	0.9462E+01	-0.9949E+01	0.1001E+02	0.9678E+01
0.03	0.3367E+00	-0.3083E+00	0.9969E+01	-0.1002E+02	0.1003E+02	0.9967E+01
0.04	0.5012E+00	-0.4055E+00	0.9999E+01	-0.1001E+02	0.1001E+02	0.9999E+01
0.05	0.6919E+00	-0.4960E+00	0.1000E+02	-0.1000E+02	0.1000E+02	0.1000E+02
0.06	0.9107E+00	-0.5764E+00	0.1000E+02	-0.1000E+02	0.1000E+02	0.1000E+02
0.07	0.1159E+01	-0.6429E+00	0.1000E+02	-0.1000E+02	0.1000E+02	0.1000E+02
0.08	0.1440E+01	-0.6913E+00	0.1000E+02	-0.1000E+02	0.1000E+02	0.1000E+02
0.09	0.1753E+01	-0.7165E+00	0.1000E+02	-0.1000E+02	0.1000E+02	0.1000E+02
0.10	0.2100E+01	-0.7128E+00	0.1000E+02	-0.1000E+02	0.1000E+02	0.1000E+02

The tracking simulation program, whose listing follows, is capable of a real-time rate of 33 frames/sec on a VAX 11/780, including the time spent simulating target motion.

```
C*****************************************************************
C                                                                C
C  Program Title: TRACK4.FOR
C
C  Function:  Demonstrate feedback tracker using synthetic
C       data.  Provide simulated motion consisting of
C       translation, magnification and rotation.
C
C  Program Author: Thomas G. Newman
C                  Department of Mathematics
C                  Texas Tech University
C                  Lubbock, Texas 79409
C
C  Notice:  Permission is herewith granted for use of these
C       programs, in whole or in part, for other than
C       personal or corporate gain.
C
C*****************************************************************
C
         COMMON /WINDOW/ IWSIZE,WIND(5,5),SWIND(5,5),COORD(2,5,5)
         COMMON /EQU/    NEQU,NUNK,A(9,9),XLAMDA(9),B(9)
         COMMON /PARMS/  TIME,XYSTEP,TSTEP,FEED(6)
         COMMON /COCHGS/ XTRANI,YTRANI,ECOSI,ESINI,
     1                  XTRANW,YTRANW,ECOSW,ESINW,
     2                  XVELI,YVELI,VMAGI,VROTI,
     3                  XVELW,YVELW,VMAGW,VROTW
100     TYPE *,'ENTER VELOCITIES AND PRESS RETURN'
        READ (5,*,END=200) XVELI,YVELI,VMAGI,VROTI
         CALL INIT
         DO 175 I=1,15
            CALL SAMPLE
            CALL DERIV
            CALL LINEQ
            CALL UPDATE
            CALL MOVER
            CALL COMPAR
 175     CONTINUE
         GO TO 100
 200     STOP
         END
C
C                          >>>>>>>>>> SUBROUTINE INIT <<<<<<<<<
C
C  This subroutine performs various initialization functions.
C
C
         SUBROUTINE INIT
         COMMON /WINDOW/ IWSIZE,WIND(5,5),SWIND(5,5),COORD(2,5,5)
         COMMON /EQU/    NEQU,NUNK,A(9,9),XLAMDA(9),B(9)
         COMMON /PARMS/  TIME,XYSTEP,TSTEP,FEED(6)
         COMMON /COCHGS/ XTRANI,YTRANI,ECOSI,ESINI,
     1                  XTRANW,YTRANW,ECOSW,ESINW,
     2                  XVELI,YVELI,VMAGI,VROTI,
     3                  XVELW,YVELW,VMAGW,VROTW
```

```
C
C        Data is to be sampled at each point of a
C        window of size IWSIZE by IWSIZE.  A linear equation
C        is to be formed at each 'interior' point, using the
C        boundary points only in the differentiation process.
C
C        The number of equations, NEQU, is therfore the number
C        of interior points, and involve NUNK unknowns, = 4
C        in this program, but changeable to 6 for GA(2).
C
C        XYSTEP and TSTEP are logical steps in space and time.
C
          IWSIZE = 5
          NEQU = (IWSIZE - 2)**2
          NUNK = 4
          XYSTEP = 0.05
          TSTEP = 0.01
C
C
C        GENERATE THE COORDINATES OF THE SAMPLE GRID RELATIVE TO
C        THE WINDOW.
C
         MID = IWSIZE/2 + 1
         DO 10 I=1,IWSIZE
              DO 15 J=1,IWSIZE
C
C             WINDOW ORIGIN AT CENTER OF SAMPLE GRID
C
                    COORD(1,I,J)=XYSTEP*FLOAT(I - MID)
                    COORD(2,I,J)=XYSTEP*FLOAT(J - MID)
C
C             WINDOW ORIGIN AT CORNER OF SAMPLE GRID
C
C                   COORD(1,I,J)=XYSTEP*FLOAT(I)
C                   COORD(2,I,J)=XYSTEP*FLOAT(J)
C
 15           CONTINUE
 10      CONTINUE
C
C        SET FEEDBACKS TO UNITY.  LOWER VALUES PRODUCE
C        SLOWER ACQUISITION BUT GREATER STABILITY.  LARGER VALUES
C        MAY SPEED AQUISITION BUT CAUSE INSTABILITY.
C
         FEED(1) = 1.0
         FEED(2) = 1.0
         FEED(3) = 1.0
         FEED(4) = 1.0
         FEED(5) = 1.0
         FEED(6) = 1.0
C
C        INITALIZE THE TRAJECTORY OF THE IMAGE AT BORE SIGHT
C
         XTRANI = 0.0
         YTRANI = 0.0
```

```
      ECOSI  = 1.0
      ESINI  = 0.0
C
C     INITALIZE THE POSITION OF THE WINDOW AT BORE SIGHT
C
      XTRANW = 0.0
      YTRANW = 0.0
      ECOSW  = 1.0
      ESINW  = 0.0
C
C     INITALIZE THE WINDOW VELOCITY TO XERO
C
      XVELW = 0.0
      YVELW = 0.0
      VMAGW = 0.0
      VROTW = 0.0
C
C     GET AN INITIAL SAMPLE FROM WINDOW
C
      TIME = 0.0
      CALL SAMPLE
C
C     TAKE THE INITIAL STEP IN TIME
C
      CALL MOVER
C
C     PRINT PAGE HEADINGS
C
      WRITE (6,1000)
      WRITE (6,1010)
 1000 FORMAT('1','TIME',5X,'XTRANI',8X,'YTRANI',8X,'ECOSI',9X,
     1      'ESINI',9X,'XVELI',9X,'YVELI',9X,'VMAGI',9X,'VROTI')
 1010 FORMAT(15X,'XTRANW',8X,'YTRANW',8X,'ECOSW',9X,
     1      'ESINW',9X,'XVELW',9X,'YVELW',9X,'VMAGW',9X,'VROTW')
C
C     PRINT THE INITIAL COMPARISON BETWEEN TRAAJECTORIES
C
      CALL COMPAR
      RETURN
      END
C
C                         >>>>>>>>> SUBROUTINE SAMPLE <<<<<<<<<<
C
C     This subroutine generates values in a rectangular
C     grid in the tracking window, saving the old values.
C
      SUBROUTINE SAMPLE
       COMMON /WINDOW/ IWSIZE,WIND(5,5),SWIND(5,5),COORD(2,5,5)
       DO 20 I=1,IWSIZE
          DO 25 J=1,IWSIZE
              X = COORD(1,I,J)
              Y = COORD(2,I,J)
              SWIND(I,J) = WIND(I,J)
               WIND(I,J) = FWIND(X,Y)
```

```
 25         CONTINUE
 20      CONTINUE
         RETURN
         END
C
C                              >>>>>>>>>> SUBROUTINE DERIV <<<<<<<<<<
C
C      This routine calculates the various derivatives needed
C      for formation of the linear system for the generalized
C      velocities.
C
         SUBROUTINE DERIV
         COMMON /WINDOW/ IWSIZE,WIND(5,5),SWIND(5,5),COORD(2,5,5)
         COMMON /EQU/    NEQU,NUNK,A(9,9),XLAMDA(9),B(9)
         COMMON /PARMS/  TIME,XYSTEP,TSTEP,FEED(6)
         SCALER = 2.0 * XYSTEP/TSTEP
         K = IWSIZE - 2
         DO 20 I=1,K
             II = I + 1
             DO 10 J=1,K
                 JJ = J + 1
                  L = (I-1)*K + J
                  X = COORD(1,II,JJ)
                  Y = COORD(2,II,JJ)
                  DX=WIND(II+1,JJ) - WIND(II-1,JJ)
                  DY=WIND(II,JJ+1) - WIND(II,JJ-1)
              A(L,1)= DX
              A(L,2)= DY
              A(L,3)= X*DX + Y*DY
              A(L,4)= X*DY - Y*DX
                B(L)= SCALER * (WIND(II,JJ) - SWIND(II,JJ))
 10          CONTINUE
 20      CONTINUE
         RETURN
         END
C
C                              >>>>>>>>>> SUBROUTINE LINEQ <<<<<<<<<<
C
C      Modified from the argument form:
C        LINEQ(M,N,A,X,B,CC)
C
         SUBROUTINE LINEQ
         COMMON /EQU/    NEQU,NUNK,A(9,9),X(9),B(9)
      INTEGER CC
C
C   SOLVE AX=B.  T HOLDS  AN UPPER TRIANGULAR MATRIX  WHILE S
C   IS WORKSPACE.  THE METHOD FACTORS A=U*T WHERE THE COLUMNS OF
C   U ARE ORTHOGANAL AND T IS TRIANGULAR.  THE RESULTING  SYSTEM
C   T*X=B' IS EASILY SOLVED BY BACK SUBSTITUTION.  ASSUME M
C   EQUATIONS AND N UNKNOWNS.  ( N <= M <= 9 )
C   THE MATRIX OF COEFFICIENTS, A IS STORED IN THE FIRST N ROWS
C   AND THE FIRST M COLUMNS OF THE 9X9 A ARRAY. THE ROUTINE
C   BRINGS IN THE WHOLE 9X9, BUT ONLY USES A(1,1) TO A(N,M)
C   (RECALL THAT FORTRAN STORES THE ARRAY COLUMN-WISE, BUT
```

```
C   ADRESSES THE ELEMENTS IN THE STANDARD ROW,COLUMN FORMAT)
C   NOTE: THE A ARRAY IS ALTERED DURING EXECUTION.
C
      DIMENSION T(9,9)
      CC=1
      M = NEQU
      N = NUNK
C   M MUST BE <= 9, AND N<=M.  CC IS A COMPLETION CODE; IF THE
C   SUBROUTINE EXECUTES PROPERLY CC WILL BE RESET TO 0 BEFORE RETURN
      DO 5 I=1,NUNK
           X(I) = 0.0
5     CONTINUE
      DO 40 I=1,N
        IF (I.EQ.1) GO TO 25
        DO 20 J=1,M
          S=0
          I1=I-1
          DO 10 K=1,I1
C                IF (T(K,K)  .LT. .0001) GO TO 5000
            S=S+A(J,K)*T(K,I)/T(K,K)
   10       CONTINUE
          A(J,I)=A(J,I)-S
   20     CONTINUE
   25   DO 40 K=I,N
          S=0
          DO 30 J=1,M
            S=S+A(J,I)*A(J,K)
   30       CONTINUE
          T(I,K)=S
   40     CONTINUE
        DO 60 I=1,N
          S=0
          DO 50 J=1,M
            S=S+A(J,I)*B(J)
   50       CONTINUE
          X(I)=S
   60     CONTINUE
        DO 80 I=1,N
          I1=N+1-I
          IF (I1.EQ.N) GO TO 75
            I2=I1+1
            DO 70 J=I2,N
              X(I1)=X(I1)-T(I1,J)*X(J)
   70         CONTINUE
C                 IF (T(I1,I1).LT..0001) GO TO 5000
   75       X(I1)=X(I1)/T(I1,I1)
   80       CONTINUE
      CC=0
      RETURN
 5000 CC=-1
C   A COMPLETION CODE OF -1 INDICATES THAT THE SUBROUTINE
C   TRIED TO DIVIDE BY 0.
      RETURN
      END
```

```
C
C                                 >>>>>>>>>> SUBROUTINE UPDATE <<<<<<<<<<
C
C       This routine updates the velocities of the window
C       following the calculation of the target velocities
C       relative to the window.  Sensitivity may be varied
C       by the feedback factors in the array FEED.
C
        SUBROUTINE UPDATE
         COMMON /WINDOW/ IWSIZE,WIND(5,5),SWIND(5,5),COORD(2,5,5)
         COMMON /EQU/    NEQU,NUNK,A(9,9),XLAMDA(9),B(9)
         COMMON /PARMS/  TIME,XYSTEP,TSTEP,FEED(6)
         COMMON /COCHGS/ XTRANI,YTRANI,ECOSI,ESINI,
     1                  XTRANW,YTRANW,ECOSW,ESINW,
     2                  XVELI,YVELI,VMAGI,VROTI,
     3                  XVELW,YVELW,VMAGW,VROTW
         DVELX = (ECOSW  * XLAMDA(1) - ESINW * XLAMDA(2)) -
     1          (XLAMDA(3) * XTRANW - XLAMDA(4) * YTRANW)
         DVELY = (ESINW * XLAMDA(1) + ECOSW * XLAMDA(2))  -
     2           (XLAMDA(4) * XTRANW + XLAMDA(3) * YTRANW)
         DMAGV = XLAMDA(3)
         DVROT = XLAMDA(4)
C
         XVELW = XVELW - FEED(1) * DVELX
         YVELW = YVELW - FEED(2) * DVELY
         VMAGW = VMAGW - FEED(3) * DMAGV
         VROTW = VROTW - FEED(4) * DVROT
C
         RETURN
         END
C
C                                 >>>>>>>>>> SUBROUTINE MOVER <<<<<<<<<<
C
C       This routine moves the window by taking a step in
C       the differential equations for the affine tranformation
C       which controls the window location.  The method used is
C       a simple Euler method.
C
C       The routine also simulates the motion of the target by
C       solving the corresponding differential equation for the
C       target.  This portion would be removed if real data were
C       being used.
C
         SUBROUTINE MOVER
         COMMON /PARMS/  TIME,XYSTEP,TSTEP,FEED(6)
         COMMON /COCHGS/ XTRANI,YTRANI,ECOSI,ESINI,
     1                  XTRANW,YTRANW,ECOSW,ESINW,
     2                  XVELI,YVELI,VMAGI,VROTI,
     3                  XVELW,YVELW,VMAGW,VROTW
         DECOS = VMAGW * ECOSW  -  VROTW * ESINW
         DESIN = VROTW * ECOSW  +  VMAGW * ESINW
         ECOSW = ECOSW + TSTEP * DECOS
         ESINW = ESINW + TSTEP * DESIN
C
```

```
            DXTRAN = XVELW + VMAGW*XTRANW - VROTW*YTRANW
            DYTRAN = YVELW + VROTW*XTRANW + VMAGW*YTRANW
            XTRANW = XTRANW + DXTRAN*TSTEP
            YTRANW = YTRANW + DYTRAN*TSTEP
C
C          The following portions simulate motion of the target.
C
            DECOS = VMAGI * ECOSI  -  VROTI * ESINI
            DESIN = VROTI * ECOSI  +  VMAGI * ESINI
            ECOSI = ECOSI + TSTEP * DECOS
            ESINI = ESINI + TSTEP * DESIN
C
            DXTRAN = XVELI + VMAGI*XTRANI - VROTI*YTRANI
            DYTRAN = YVELI + VROTI*XTRANI + VMAGI*YTRANI
            XTRANI = XTRANI + DXTRAN*TSTEP
            YTRANI = YTRANI + DYTRAN*TSTEP
C
C          Increment time.
C
           TIME = TIME + TSTEP
C
           RETURN
           END
C
C                                  >>>>>>>>>> SUBROUTINE COMPAR <<<<<<<<<<
C
C          This routine produces printed output for evaluation
C          purposes, and is therefore ancillary to the operation
C          of the tracker.
C
           SUBROUTINE COMPAR
            COMMON /PARMS/  TIME,XYSTEP,TSTEP,FEED(6)
            COMMON /COCHGS/ XTRANI,YTRANI,ECOSI,ESINI,
     1                     XTRANW,YTRANW,ECOSW,ESINW,
     2                     XVELI,YVELI,VMAGI,VROTI,
     3                     XVELW,YVELW,VMAGW,VROTW
            WRITE(6,2000)TIME,XTRANI,YTRANI,ECOSI,ESINI,XVELI,YVELI,VMAGI,
     1           VROTI
 2000       FORMAT(2X,F3.2,8(3X,E11.4))
            WRITE(6,2010)XTRANW,YTRANW,ECOSW,ESINW,XVELW,YVELW,VMAGW,
     1           VROTW
 2010       FORMAT(10X,8(3X,E11.4))
            RETURN
            END
C
C                                  >>>>>>>>>> FUNCTION FWIND <<<<<<<<<<
C
C          This function returns a value at the point (x,y)
C          in the window.  It first maps to the corresponding
C          point in absolute image coordinates and calls for
C          image value at that point (see FIMAGE below).
C
            FUNCTION FWIND(X,Y)
            COMMON /COCHGS/ XTRANI,YTRANI,ECOSI,ESINI,
```

```
     1                    XTRANW,YTRANW,ECOSW,ESINW,
     2                    XVELI,YVELI,VMAGI,VROTI,
     3                    XVELW,YVELW,VMAGW,VROTW
C
C
C
      XIMAGE = XTRANW + ECOSW*X - ESINW*Y
      YIMAGE = YTRANW + ESINW*X + ECOSW*Y
      FWIND  = FIMAGE(XIMAGE,YIMAGE)
      RETURN
      END
C
C                         >>>>>>>>>> FUNCTION FIMAGE <<<<<<<<<<
C
C    This function returns the value at point (x,y) in the
C    absolute image coordinate system.  It maps to target
C    coordinates and calls for the value at the corresponding
C    point on the target (see FOBJ below).
C
      FUNCTION FIMAGE(X,Y)
      COMMON /COCHGS/ XTRANI,YTRANI,ECOSI,ESINI,
     1                    XTRANW,YTRANW,ECOSW,ESINW,
     2                    XVELI,YVELI,VMAGI,VROTI,
     3                    XVELW,YVELW,VMAGW,VROTW
C
C
      DET = ECOSI**2 + ESINI**2
       DX = X - XTRANI
       DY = Y - YTRANI
C
C
      XOBJ= (  ECOSI*DX + ESINI*DY)/DET
      YOBJ= ( -ESINI*DX + ECOSI*DY)/DET
      FIMAGE = FOBJ(XOBJ,YOBJ)
      RETURN
      END
C
C                         >>>>>>>>>> FUNCTION FOBJ <<<<<<<<<<
C
C    This function returns the gray value at a point (x,y)
C    in target coordinates.  For actual tracking, this
C    routine would be replaced by one which retrieves from
C    the image database. In the simulation, however, the
C    routine merely returns a synthetic value generated
C    from an expression.
C
      FUNCTION FOBJ(X,Y)
      FOBJ = 1.0 + 10.0*X -5.0*Y + 20.0*X*Y
      RETURN
      END
```

APPENDIX B

Tracking With Differential Forms

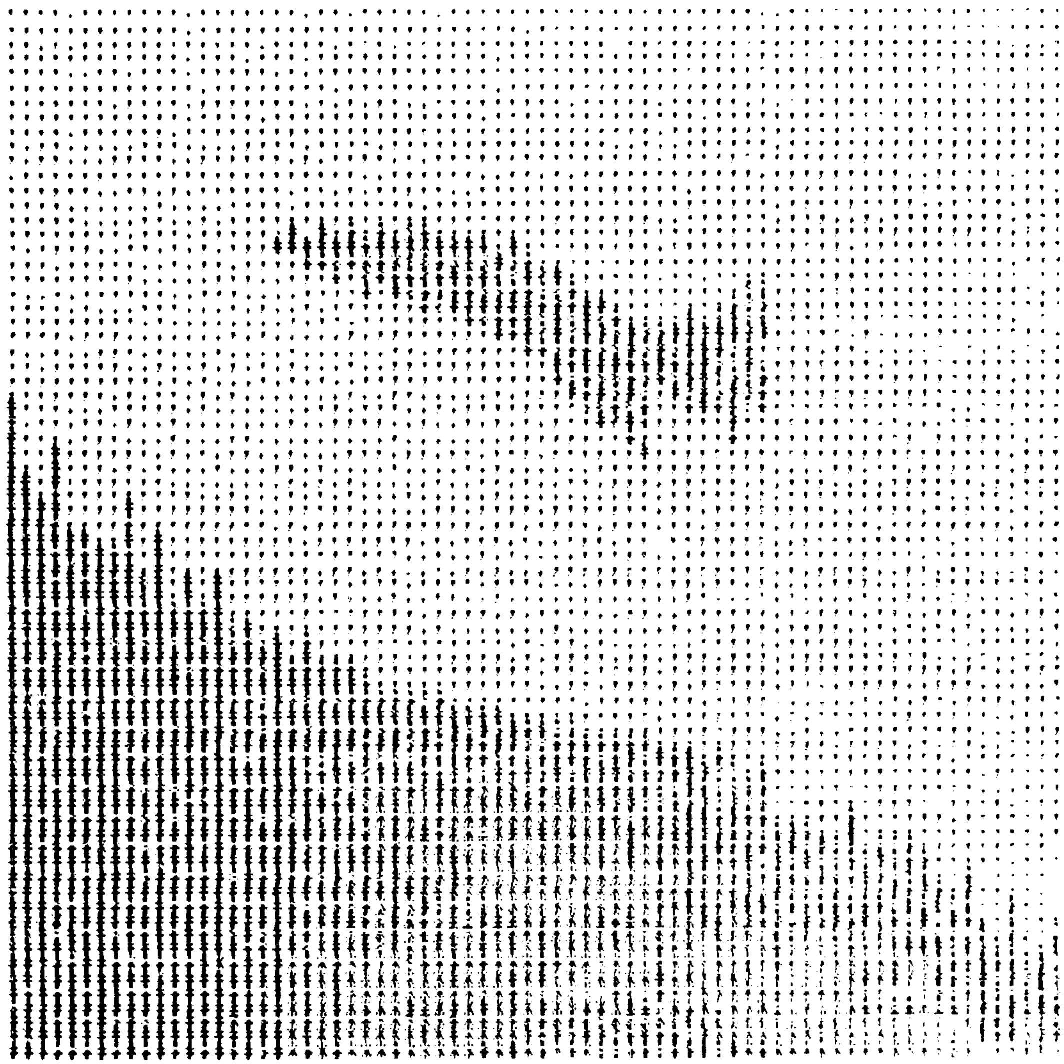
A tracking program was developed which utilized the theory presented in Section III. In order to test this program, digitized video images were obtained from the Advanced Technology Office, Instrumentation Directorate, White Sands Missile Range. One such image is shown in Figure B-1. A sequence of test images was prepared from this data by shifting to inject additional motion. Sections of the first six frames are shown in the left hand column of Figure B-2.

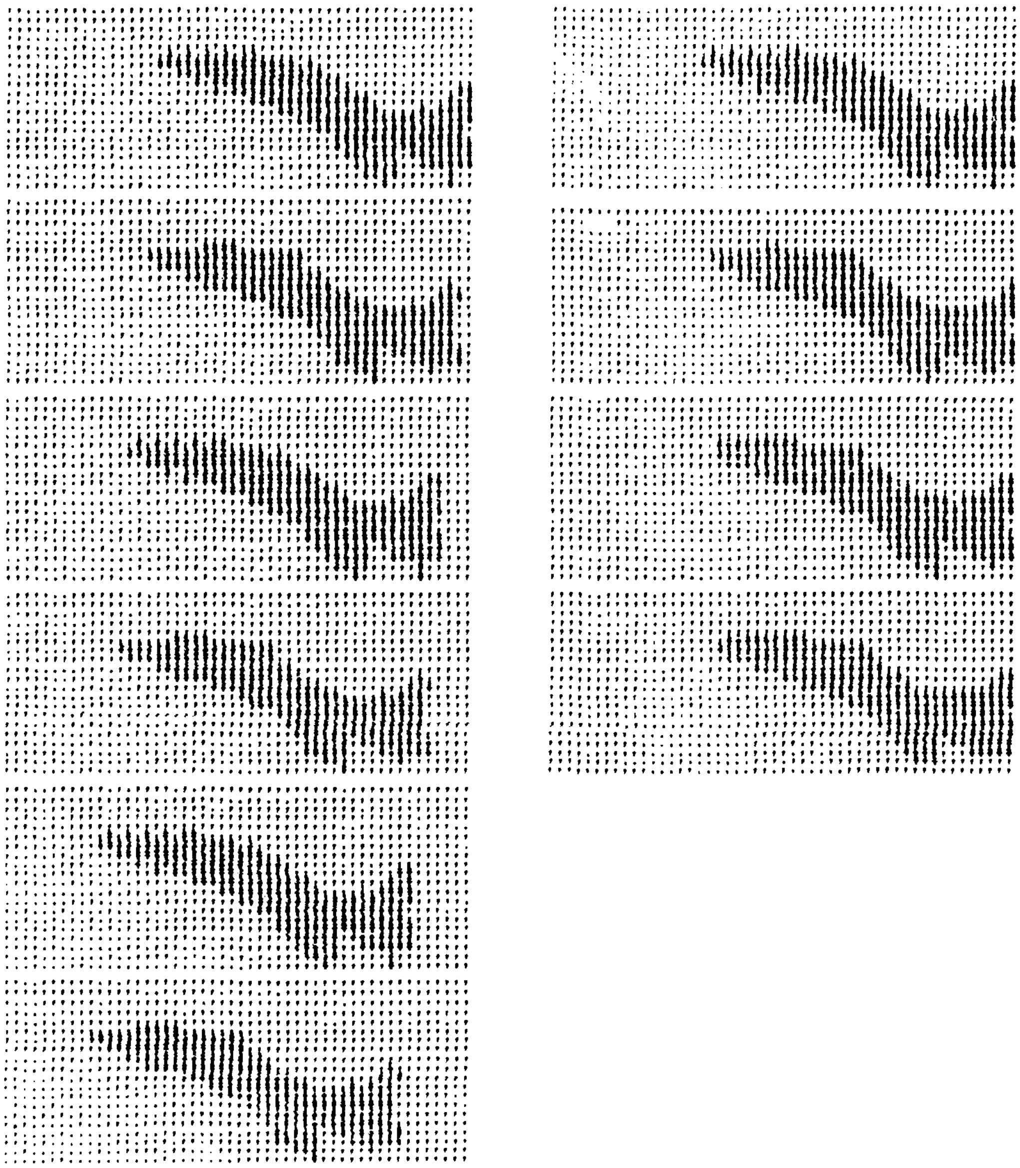
Parameters in the tracking program were set to use a 3x3 window in 3 consecutive frames to form a 3x3x3 rectangle in space. Since the program uses 9 such windows, the actual track gate consisted of 9x9x3 points.

The right hand column of Figure B-2 shows the output frames obtained by the tracker. Observe that the output lags the input by the depth of the track gate. This is why there are fewer outputs frames than input frames. We see that the target was acquired immediately, and successfully tracked over the full sequence of frames.

The computation rate was about 10 frames per second on a VAX 11/780. However, the program is written to allow selection of window sizes and could be streamlined a great deal.

Although the results shown in Figure B-2 are impressive, we hasten to point out that the motion is mostly translation, and our attempts to test the algorithm on a wide range of motions have been frustrated by availability of data. Further work is ongoing, and refinements to the tracking program are expected to be forthcoming.





```
C       PROGRAMMER : DONNA K. TERRAL
C                    DEPARTMENT OF MATHEMATICS
C                    TEXAS TECH UNIVERSITY
C
C       PERMISSION IS HEREWITH GRANTED TO UTILIZE THIS PROGRAM FOR
C       OTHER THAN PERSONAL OR CORPORATIVE GAIN.
C
C
C ********************************************************************
C *************************** MAIN PROGRAM ***************************
C ********************************************************************
C       PURPOSE: GIVEN A SET OF IMAGES, TRACK A TRAGET BY USING
C       INTEGRATION TO DETERMINE THE MOTIONS (TRANSLATION,ROTATION,
C       MAGNIFICATION) OF THE TARGET.
C ********************************************************************
C
C      * COMMON DECLARATIONS *
C
        PARAMETER (IW=9,JW=9,KW=3,IB=3,JB=3,KB=3,ISIZE=128,JSIZE=128)
C
        COMMON /WINDOW/ WIND(IW,JW,KW),CORRD(ISIZE,JSIZE,2)
        INTEGER WIND
        COMMON /ORIGIN/ I0,J0,K0,X0,Y0
        INTEGER I0,J0,K0,X0,Y0
        COMMON /BUFFER/ BUF(IB,JB,KB)
        INTEGER BUF
        COMMON /CORNER/ X1,X2,Y1,Y2,T1,T2
        INTEGER X1,X2,Y1,Y2,T1,T2
        COMMON /WEIGHT/ W(IB,JB,KB),WX(JB,KB),WY(IB,KB),WT(IB,JB)
        INTEGER W,WX,WY,WT
        COMMON /COCHGS/ XVELI,YVELI,VROTI,VMAGI,TIME,TSTEP
        REAL XVELI,YVELI,VROTI,VMAGI,TIME,TSTEP
        COMMON /PARMS/ ROW,COL,PROW,PCOL,NUMROW,NUMCOL,WTEST
        INTEGER ROW,COL,PROW,PCOL,NUMROW,NUMCOL,WTEST
        COMMON /IMAGE/ IMAGE(ISIZE,JSIZE)
        INTEGER*2 IMAGE
        COMMON /EQU/ ALPHA(4),BETA,COEFF(9,4),VECTOR(9),SOLN(4),FEED(4)
        INTEGER ALPHA,BETA
C
C      * MAIN VARIABLES *
        INTEGER I1,J1,K1
        INTEGER CC,COUNT
        DATA COUNT,CC,K1 /1,1,1/
C
        CALL INIT
C
C       ***  MAIN LOOP  ****
        DO 100 LOOP = 1,4
C
C          ALL OR A PORTION OF THE INTEGRATION RESULTS CAN BE USED
           SOLN(1) = FEED(1) * SOLN(1)
           SOLN(2) = FEED(2) * SOLN(2)
           SOLN(3) = FEED(3) * SOLN(3)
           SOLN(4) = FEED(4) * SOLN(4)
```

```
C
            CALL WINDOW (LOOP)
C
            CALL PRINT
C
C           *** LOOP COMPUTES MOTION USED TO MOVE WINDOW ***
            DO 1 J1 = 1,JW,JB
               DO 2 I1 = 1,IW,IB
                  CALL GETBUF (I1,J1,K1)
                  IF (LOOP .NE. 1) GO TO 10
                  CALL BLDW
 10               CALL GETEQU
                  CALL MATRIX (COUNT)
                  COUNT = COUNT + 1
 2             CONTINUE
 1          CONTINUE
            CALL LINEQ (COEFF,SOLN,VECTOR,9,4,CC)
            COUNT = 1
C           **** END MOTION LOOP ***
C
            TIME = TIME + TSTEP
 100     CONTINUE
C        ***  END MAIN LOOP  ***
C
         STOP
         END
C
C *******************************************************************
C *******************************************************************
C        SUBROUTINE GETBUF FILLS A BUFFER ARRAY WHICH WILL CONTAIN THE
C        DATA POINTS FORMING THE CUBE TO BE USED IN SUBROUTINE GETEQU.
C        THE POSITION IN WHICH TO BEGIN IS PASSED THRU THE ARGUMENTS.
C        <BUF(1,1,1)=WIND(I1,J1,K1)>
C *******************************************************************
C
         SUBROUTINE GETBUF (I1,J1,K1)
C
C        * ARGUMENTS *
         INTEGER I1,J1,K1
C
C        * COMMON DECLARATIONS *
C
         PARAMETER (IW=9,JW=9,KW=3,IB=3,JB=3,KB=3,ISIZE=128,JSIZE=128)
C
         COMMON /WINDOW/ WIND(IW,JW,KW),CORRD(ISIZE,JSIZE,2)
         INTEGER WIND
         COMMON /ORIGIN/ I0,J0,K0,X0,Y0
         INTEGER I0,J0,K0,X0,Y0
         COMMON /BUFFER/ BUF(IB,JB,KB)
         INTEGER BUF
         COMMON /CORNER/ X1,X2,Y1,Y2,T1,T2
         INTEGER X1,X2,Y1,Y2,T1,T2
C
C        * LOCAL VARAIBLES *
```

```
      INTEGER X,Y,T
C
      X1 = I1 - I0
      Y1 = J1 - J0
      T1 = K1 - K0
      X2 = X1 + IB - 1
      Y2 = Y1 + JB - 1
      T2 = T1 + KB - 1
C
      DO 30 I = 1,IB
        DO 30 J = 1,JB
          DO 30 K = 1,KB
            X = I1 + I - 1
            Y = J1 + J - 1
            T = K1 + K - 1
            BUF(I,J,K) = WIND(X,Y,T)
 30   CONTINUE
      RETURN
      END
C
C ******************************************************************
C ******************************************************************
C     SUBROUTINE MATRIX TAKES THE ALPHAS AND BETA FROM THE SUBROUTINE
C     GETEQU AND PUTS THEM IN THE FORM AX=B WHERE LOOPS OF GETEQU
C     FORM THE 2-DIMENSIONAL ARRAY A AND THE VECTOR B.
C     ONCE AX=B IS FORMED LINEQ IS USED TO SOLVE FOR X.
C
C     IN MATRIX, COEFF(9,4) IS ARRAY A AND VECTOR(9) IS ARRAY B
C ******************************************************************
C
      SUBROUTINE MATRIX (COUNT)
C
C     * ARGUMENTS *
      INTEGER COUNT
C
C     * COMMON DECLARATIONS *
      COMMON /EQU/ ALPHA(4),BETA,COEFF(9,4),VECTOR(9),SOLN(4),FEED(4)
      INTEGER ALPHA,BETA
C
      COEFF(COUNT,1) = FLOAT(ALPHA(1))
      COEFF(COUNT,2) = FLOAT(ALPHA(2))
      COEFF(COUNT,3) = FLOAT(ALPHA(3))
      COEFF(COUNT,4) = FLOAT(ALPHA(4))
      VECTOR(COUNT) = FLOAT(BETA)
      RETURN
      END
C
C *********************************************************************
C *********************************************************************
C     SUBROUTINE GETEQU
C     COMPUTES CONSTANTS ALPHA(1) THRU ALPHA(4) AND BETA IN THE
C     EQUATION ALPHA(1) * LAMBDA(1) + ALPHA(2) * LAMBDA(2) +
C     ALPHA(3) * LAMBDA(3) + ALPHA(4) * LAMBDA(4) = BETA, WHERE
C     ALPHA(1) THRU ALPHA(4) AND BETA ARE FORMED FROM SURFACE AND
```

```
C       VOLUME INTEGALS OVER A CUBE INDEXED BY X1,X2,Y1,Y2,T1,T2.
C       THESE INTEGALS ARE NUMERICALLY INTEGRATED USING THE TRAPEZIOD
C       RULE.
C
C       LET:
C         FXYT REPRESENT THE VOLUME INTEGAL OVER THE CUBE
C         FXY1 REPRESENT THE SURFACE INTEGAL OVER THE FACE XY @ T=1
C         FXY2     '      '      '         '      '    '  FACE XY @ T=2
C         FYT1     '      '      '         '      '    '  FACE YT @ X=1
C         FYT2     '      '      '         '      '    '  FACE YT @ X=2
C         FTX1     '      '      '         '      '    '  FACE TX @ Y=1
C         FTX2     '      '      '         '      '    '  FACE TX @ Y=2
C         YFYT1 REPRESENT THE INTEGAL OVER Y * (FACE YT) @ X=1
C         YFYT2    '       '       '      '  Y * (FACE YT) @ X=2
C         XFTX1    '       '       '      '  X * (FACE TX) @ Y=1
C         XFTX2    '       '       '      '  X * (FACE TX) @ Y=2
C
C       THEN:
C         ALPHA(1) = FYT2 - FYT1
C         ALPHA(2) = FTX2 - FTX1
C         ALPHA(3) = X2*FYT2 - X1*FYT1 - 2*FXYT + Y2*FTX2 - Y1*FTX1
C         ALPHA(4) = YFYT1 - YFYT2 + XFTX2 - XFTX1
C         BETA = FXY1 - FXY2
C *********************************************************************
C
        SUBROUTINE GETEQU
C
C       * COMMON DECLARATIONS *
C
        PARAMETER (IW=9,JW=9,KW=3,IB=3,JB=3,KB=3,ISIZE=128,JSIZE=128)
C
        COMMON /BUFFER/ BUF(IB,JB,KB)
        INTEGER BUF
        COMMON /CORNER/ X1,X2,Y1,Y2,T1,T2
        INTEGER X1,X2,Y1,Y2,T1,T2
        COMMON /WEIGHT/ W(IB,JB,KB),WX(JB,KB),WY(IB,KB),WT(IB,JB)
        INTEGER W,WX,WY,WT
        COMMON /EQU/ ALPHA(4),BETA,COEFF(9,4),VECTOR(9),SOLN(4),FEED(4)
        INTEGER ALPHA,BETA
C
C       * LOCAL VARIABLES *
        INTEGER FXY1,FXY2,FYT1,FYT2,FTX1,FTX2,YFYT1,YFYT2,XFTX1,XFTX2,FXYT
C
        FXYT = 0
        FXY1 = 0
        FXY2 = 0
        FYT1 = 0
        FYT2 = 0
        FTX1 = 0
        FTX2 = 0
        YFYT1 = 0
        YFYT2 = 0
        XFTX1 = 0
        XFTX2 = 0
```

```
C
         DO 50 I = 1,IB
           DO 50 J = 1,JB
             FXY1 = FXY1 + BUF(I,J,1) * WT(I,J)
             FXY2 = FXY2 + BUF(I,J,KB) * WT(I,J)
 50      CONTINUE
         DO 60 J = 1,JB
           DO 60 K = 1,KB
             FYT1 = FYT1 + BUF(1,J,K) * WX(J,K)
             FYT2 = FYT2 + BUF(IB,J,K) * WX(J,K)
             YFYT1 = YFYT1 + (J + Y1 - 1) * BUF(1,J,K) * WX(J,K)
             YFYT2 = YFYT2 + (J + Y1 - 1) * BUF(IB,J,K) * WX(J,K)
 60      CONTINUE
         DO 70 I = 1,IB
           DO 70 K = 1,KB
             FTX1 = FTX1 + BUF(I,1,K) * WY(I,K)
             FTX2 = FTX2 + BUF(I,JB,K) * WY(I,K)
             XFTX1 = XFTX1 + (I + X1 - 1) * BUF(I,1,K) * WY(I,K)
             XFTX2 = XFTX2 + (I + X1 - 1) * BUF(I,JB,K) * WY(I,K)
 70      CONTINUE
         DO 80 I = 1,IB
           DO 80 J = 1,JB
             DO 80 K = 1,KB
               FXYT = FXYT + BUF(I,J,K) * W(I,J,K)
 80      CONTINUE
C
         ALPHA(1) = FYT2 - FYT1
         ALPHA(2) = FTX2 - FTX1
         ALPHA(3) = X2 * FYT2 - X1 * FYT1 - FXYT + Y2 * FTX2 - Y1 * FTX1
         ALPHA(4) = YFYT1 - YFYT2 + XFTX2 -XFTX1
         BETA = FXY1 - FXY2
C
         RETURN
         END
C
C ********************************************************************
C ********************************************************************
C        SUBROUTINE BLDW BUILDS THE WEIGHING ARRAYS FOR THE
C        TRAPEZIOD RULE USED IN THE SUBROUTINE GETEQU. THESE ARRAYS
C        ARE PASSED TO GETEQU THRU A COMMON BLOCK.
C ********************************************************************
C
         SUBROUTINE BLDW
C
C       * COMMON DECLARATIONS *
C
         PARAMETER (IW=9,JW=9,KW=3,IB=3,JB=3,KB=3,ISIZE=128,JSIZE=128)
C
         COMMON /WEIGHT/ W(IB,JB,KB),WX(JB,KB),WY(IB,KB),WT(IB,JB)
         INTEGER W,WX,WY,WT
C
C       LOCAL VARAIBLES
           INTEGER X,Y,T
C
```

```
      X = IB - 1
      Y = JB - 1
      T = KB - 1
C
      DO 40 I = 2,X
        DO 40 J = 2,Y
          DO 40 K = 2,T
            W(I,1,1) = 2
            W(I,JB,1) = 2
            W(1,J,1) = 2
            W(IB,J,1) = 2
            W(1,JB,K) = 2
            W(IB,1,K) = 2
            W(I,1,KB) = 2
            W(1,J,KB) = 2
            W(IB,J,KB) = 2
            W(I,JB,KB) = 2
            W(IB,JB,K) = 2
            W(1,1,K) = 2
            W(I,J,K) = 8
            W(1,J,K) = 4
            W(I,1,K) = 4
            W(I,J,1) = 4
            W(I,JB,K) = 4
            W(IB,J,K) = 4
            W(I,J,KB) = 4
            WY(I,1) = 2
            WY(1,K) = 2
            WY(I,K) = 4
            WY(I,KB) = 2
            WY(IB,K) = 2
            WT(I,1) = 2
            WT(1,J) = 2
            WT(I,J) = 4
            WT(I,JB) = 2
            WT(IB,J) = 2
            WX(JB,K) = 2
            WX(J,KB) = 2
            WX(1,K) = 2
            WX(J,1) = 2
            WX(J,K) = 4
   40 CONTINUE
      W(1,1,1) = 1
      W(1,JB,1) = 1
      W(1,JB,KB) = 1
      W(1,1,KB ) = 1
      W(IB,1,1) = 1
      W(IB,JB,1) = 1
      W(IB,1,KB) = 1
      W(IB,JB,KB) = 1
      WX(1,1) = 1
      WX(1,KB) = 1
      WX(JB,1) = 1
      WX(JB,KB) = 1
```

```
      WY(1,1) = 1
      WY(1,KB) = 1
      WY(IB,1) = 1
      WY(IB,KB) = 1
      WT(1,1) = 1
      WT(1,JB) = 1
      WT(IB,1) = 1
      WT(IB,JB) = 1
      RETURN
      END
C
C ***********************************************************************
C ***********************************************************************
C     SUBROUTINE PRINT WRITES TO UNIT = 66 THE MOTION PARAMETERS
C     OF THE IMAGE AND THE WINDOW AT EACH TIME STEP. THE PIXEL
C     VALUES IN THE WINDOW MAY ALSO BE PRINTED IF NEEDED.
C ***********************************************************************
C
      SUBROUTINE PRINT
C
C     * COMMON DECLARATIONS *
C
      PARAMETER (IW=9,JW=9,KW=3,IB=3,JB=3,KB=3,ISIZE=128,JSIZE=128)
C
      COMMON /WINDOW/ WIND(IW,JW,KW),CORRD(ISIZE,JSIZE,2)
      INTEGER WIND
      COMMON /ORIGIN/ I0,J0,K0,X0,Y0
      INTEGER I0,J0,K0,X0,Y0
      COMMON /COCHGS/ XVELI,YVELI,VROTI,VMAGI,TIME,TSTEP
      REAL XVELI,YVELI,VROTI,VMAGI,TIME,TSTEP
      COMMON /PARMS/ ROW,COL,PROW,PCOL,NUMROW,NUMCOL,WTEST
      INTEGER ROW,COL,PROW,PCOL,NUMROW,NUMCOL,WTEST
      COMMON /EQU/ ALPHA(4),BETA,COEFF(9,4),VECTOR(9),SOLN(4),FEED(4)
      INTEGER ALPHA,BETA
C
C     * LOCAL VARIABLES *
      REAL XTRANW,YTRANW,ROTW,MAGW,XVELW,YVELW,VROTW,VMAGW,XTRANI,
     &          MAGI,ROTI,YTRANI
C
      DATA XTRANW,YTRANW,ROTW,MAGW /0,0,0,0/
C
      XTRANI = XVELI*TIME
      YTRANI = YVELI*TIME
      MAGI = VMAGI*TIME
      ROTI = VROTI*TIME
C
      XTRANW = SOLN(1) + XTRANW
      YTRANW = SOLN(2) + YTRANW
      MAGW = SOLN(3) + MAGW
      ROTW = SOLN(4) + ROTW
C
      XVELW = SOLN(1)*(1/TSTEP)
      YVELW = SOLN(2)*(1/TSTEP)
      VMAGW = SOLN(3)*(1/TSTEP)
```

```
            VROTW = SOLN(4)*(1/TSTEP)
C
            IF (WTEST .EQ. 1) THEN
               ITEMP = COL - PCOL + (IW+1)/2
               JTEMP = ROW - PROW + (JW+1)/2
               WRITE(66,30) CORRD(ITEMP,JTEMP,1),CORRD(ITEMP,JTEMP,2)
  30           FORMAT (1X,'CENTER OF WINDOW AT (', F9.5',',F9.5,')',/)
C
               WRITE (66,36)
  36           FORMAT (//1X,'WINDOW VALUES',/)
               DO 4 K = 1,KW
                 DO 5 J = 1,JW
                    WRITE (66,6) (WIND(I,J,K),I=1,IW)
  6                 FORMAT (1X,<IW>(I4))
  5              CONTINUE
                 WRITE (66,8)
  8              FORMAT (/)
  4            CONTINUE
C
               WRITE (66,10) TIME,XTRANI,YTRANI,MAGI,ROTI,XVELI,YVELI,
     &                       VMAGI,VROTI
  10           FORMAT (1X,F5.4,8(3X,E11.4),/)
               WRITE (66,11)
  11           FORMAT(10X,'XTRANW',8X,'YTRANW',8X,'MAGW',10X,'ROTW',
     &               10X,'XVELW',9X,'YVELW',9X,'VMAGW',9X,'VROTW')
               WRITE (66,12) XTRANW,YTRANW,MAGW,ROTW,XVELW,YVELW,VMAGW,VROTW
  12           FORMAT (6X,8(3X,E11.4),/)
               WRITE (66,13)
  13           FORMAT('1')
C
            ELSE
               WRITE (66,14) TIME,XTRANI,YTRANI,MAGI,ROTI,XVELI,YVELI,
     &                       VMAGI,VROTI
  14           FORMAT(1X,F5.4,8(3X,E11.4))
               WRITE(66,15) XTRANW,YTRANW,MAGW,ROTW,XVELW,YVELW,VMAGW,VROTW
  15           FORMAT(11X,8(3X,E11.4),/)
            ENDIF
            RETURN
            END
C
C ***********************************************************************
C ***********************************************************************
C           SUBROUTINE INIT IS USED TO INITIALIZE PARAMETERS.
C **********************************************************************
C
            SUBROUTINE INIT
C
C           * COMMON DECLARATIONS *
            PARAMETER (IW=9,JW=9,KW=3,IB=3,JB=3,KB=3,ISIZE=128,JSIZE=128)
C
            COMMON /WINDOW/ WIND(IW,JW,KW),CORRD(ISIZE,JSIZE,2)
            INTEGER WIND
            COMMON /ORIGIN/ IO,JO,KO,XO,YO
            INTEGER IO,JO,KO,XO,YO
```

```
      COMMON /PARMS/ ROW,COL,PROW,PCOL,NUMROW,NUMCOL,WTEST
      INTEGER ROW,COL,PROW,PCOL,NUMROW,NUMCOL,WTEST
      COMMON /COCHGS/ XVELI,YVELI,VROTI,VMAGI,TIME,TSTEP
      COMMON /EQU/ ALPHA(4),BETA,COEFF(9,4),VECTOR(9),SOLN(4),FEED(4)
      INTEGER ALPHA,BETA
C
C     THE PIXEL VALUE AT (COL,ROW) IS PUT INTO WIND(1,1,1)
      WRITE (6,20)
 20   FORMAT (1X,'BEGIN WINDOW')
      WRITE (6,50)
 50   FORMAT (1X,'ROW      COL')
      READ (5,*) ROW,COL
C
C     THE MOTION TO BE TRACKED
      WRITE (6,30)
 30   FORMAT (1X,'INPUT:XVELI,YVELI,VROTI/PI,VMAGI')
      READ (5,*) XVELI,YVELI,VROTI,VMAGI
C
      PROW = 1
      PCOL = 1
      NUMROW = 128
      NUMCOL = 128
C
      X0 = COL + (IW-1)/2
      Y0 = ROW + (JW-1)/2
C
      DO 60 I=1,NUMCOL
         DO 70 J=1,NUMROW
            CORRD(I,J,1) = I
            CORRD(I,J,2) = J
 70      CONTINUE
 60   CONTINUE
C
      TSTEP = 0.033
      TIME = 2*TSTEP
      PI = 3.14159
      VROTI = VROTI*PI
C
      I0 = (IW+1)/2
      J0 = (JW+1)/2
      K0 = (KW+1)/2
C
      DO 10 I=1,4
         SOLN(I) = 0.0
 10   CONTINUE
C
      FEED(1) = 1.0
      FEED(2) = 1.0
      FEED(3) = 1.0
      FEED(4) = 1.0
C
      WRITE (6,40)
 40   FORMAT (1X,'INPUT 1 cr. TO WRITE WINDOW OR 0 cr. TO SKIP')
      READ (5,*) WTEST
```

```
      IF (WTEST .EQ. 0) THEN
         WRITE (66,5) IB
5        FORMAT (1X,'WINDOW SIZE = ',I2,//)
         WRITE (66,3)
3        FORMAT(1X,'TIME',5X,'XTRANI',8X,'YTRANI',8X,'MAGI',10X,'ROTI'
     &          ,10X,'XVELI',9X,'YVELI',9X,'VMAGI',9X,'VROTI')
         WRITE (66,4)
4        FORMAT(15X,'XTRANW',8X,'YTRANW',8X,'MAGW',10X,'ROTW',
     &          10X,'XVELW',9X,'YVELW',9X,'VMAGW',9X,'VROTW',/)
      ENDIF
C
      RETURN
      END
C
C ********************************************************************
C ********************************************************************
C     SUBROUTINE WINDOW (1) USES THE INTEGRATION RESULTS TO MOVE THE
C     WINDOW IN ORDER TO TRACK THE TARGET AND (2) PERFORMS THE MOTION
C     ON THE 1ST IMAGE USED TO GET THE WINDOW AND WRITES THE RESULT
C     TO UNIT = M + 15.
C *********************************************************************
C
      SUBROUTINE WINDOW (M)
C
C     * COMMON DECLARATIONS *
C
      PARAMETER (IW=9,JW=9,KW=3,IB=3,JB=3,KB=3,ISIZE=128,JSIZE=128)
C
      COMMON /WINDOW/ WIND(IW,JW,KW),CORRD(ISIZE,JSIZE,2)
      INTEGER WIND
      COMMON /EQU/ ALPHA(4),BETA,COEFF(9,4),VECTOR(9),SOLN(4),FEED(4)
      INTEGER ALPHA,BETA
      COMMON /IMAGE/ IMAGE(ISIZE,JSIZE)
      INTEGER*2 IMAGE
      COMMON /BUFFER/ BUF(IB,JB,KB)
      INTEGER BUF
      COMMON /PARMS/ ROW,COL,PROW,PCOL,NUMROW,NUMCOL,WTEST
      INTEGER ROW,COL,PROW,PCOL,NUMROW,NUMCOL,WTEST
C
C     * LOCAL VARIABLES *
      LOGICAL*1 BIMAGE(ISIZE),BLINE(2*ISIZE,JSIZE)
      CHARACTER*(ISIZE) IMAGELINE
      INTEGER*2 NEWIM(ISIZE,JSIZE)
      REAL ECOSW,ESINW,XW,YW
      EQUIVALENCE (BIMAGE,IMAGELINE),(BLINE,NEWIM)
C
C     AFFINE TRANSFORMATION TO MOVE WINDOW
C
      ITEMP = COL - PCOL + (IB+1)/2
      JTEMP = ROW - PROW + (JB+1)/2
      XW = CORRD(ITEMP,JTEMP,1)
      YW = CORRD(ITEMP,JTEMP,2)
      ECOSW = EXP(SOLN(3)) * COS(SOLN(4))
      ESINW = EXP(SOLN(3)) * SIN(SOLN(4))
```

```
      C
              DO 10 J=1,NUMROW
                 DO 20 I=1,NUMCOL
                    CORRD(I,J,1)=ECOSW*(CORRD(I,J,1)-XW)-ESINW*(CORRD(I,J,2)
           &              -YW)+ECOSW*SOLN(1)-ESINW*SOLN(2)
                    CORRD(I,J,2)=ESINW*(CORRD(I,J,1)-XW)+ECOSW*(CORRD(I,J,2)
           &              -YW)+ESINW*SOLN(1)+ECOSW*SOLN(2)
                    CORRD(I,J,1) = CORRD(I,J,1) + XW
                    CORRD(I,J,2) = YW + CORRD(I,J,2)
       20        CONTINUE
       10     CONTINUE
      C
      C       OPEN A IMAGE FILE AND FILL PIXEL VALUES INTO THE ARRAY IMAGE
      C
              DO 30 K=1,KW
                 KCOUNT = M + K + 9
                 OPEN(UNIT=KCOUNT,STATUS='OLD',ACCESS='DIRECT',
           &          RECORDTYPE='FIXED',READONLY)
      C
                 DO 80 I=1,ISIZE
                    DO 90 J=1,JSIZE
                       IMAGE(I,J) = 0
       90           CONTINUE
       80        CONTINUE
      C
                 DO 40 J=1,NUMROW
                    READ (KCOUNT'J) IMAGELINE
                    DO 50 I=1,ISIZE
                       IMAGE(I,J) = BIMAGE(I) .AND. 255
       50           CONTINUE
       40        CONTINUE
      C
      C       USE THE TRANSFORMATION TO TRACK; STORE THE RESULT IN NEWIM
      C
                 DO 60 J=1,NUMROW
                    DO 70 I=1,NUMCOL
                       IF (CORRD(I,J,1) .LT. 1 .OR. CORRD(I,J,1) .GT. ISIZE
           &           .OR. CORRD(I,J,2) .LT. 1 .OR. CORRD(I,J,2) .GT. JSIZE)
           &           THEN
                           NEWIM(I,J) = 0
                       ELSE
                           NEWIM(I,J) = IBILIN(CORRD(I,J,1),CORRD(I,J,2))
                       ENDIF
       70           CONTINUE
       60        CONTINUE
      C
                 CLOSE (UNIT=KCOUNT)
      C
      C       FILL WINDOW WITH NEW PIXEL VALUES
      C
                 DO 110 I=1,IW
                    DO 120 J=1,JW
                       ITEMP = COL - PCOL + I
                       JTEMP = ROW - PROW + J
```

```
                     WIND(I,J,K) = NEWIM(ITEMP,JTEMP)
  120             CONTINUE
  110         CONTINUE
C
C        WRITE TRACKED IMAGE INTO A FILE
C
              IF (K .EQ. 1) THEN
                 OPEN(UNIT=KCOUNT+15,STATUS='NEW',ACCESS='DIRECT',
     &                  RECORDTYPE='FIXED',RECL=NUMCOL/4,BLOCKSIZE=NUMCOL)
                 DO 130 J=1,NUMROW
                    DO 140 I=1,NUMCOL
                       BIMAGE(I) = BLINE(I*2-1,J)
  140               CONTINUE
                    WRITE (KCOUNT+15'J)IMAGELINE
  130            CONTINUE
                 CLOSE (UNIT=KCOUNT+15)
                 ENDIF
C
  30      CONTINUE
          RETURN
          END
C
C *********************************************************************
C ********************************************************************
C
        SUBROUTINE LINEQ(A,X,B,M,N,CC)
        INTEGER CC
C
C   SOLVE AX=B.  T HOLDS  AN UPPER TRIANGULAR MATRIX  WHILE S
C   IS WORKSPACE.  THE METHOD FACTORS A=U*T WHERE THE COLUMNS OF
C   U ARE ORTHOGANAL AND T IS TRIANGULAR.  THE RESULTING  SYSTEM
C   T*X=B' IS EASILY SOLVED BY BACK SUBSTITUTION.  ASSUME M
C   EQUATIONS AND N UNKNOWNS.  ( N <= M <= 9 )
C   THE MATRIX OF COEFFICIENTS, A IS STORED IN THE FIRST N ROWS
C   AND THE FIRST M COLUMNS OF THE 9X9 A ARRAY. THE ROUTINE
C   BRINGS IN THE WHOLE 9X9, BUT ONLY USES A(1,1) TO A(N,M)
C   (RECALL THAT FORTRAN STORES THE ARRAY COLUMN-WISE, BUT
C   ADRESSES THE ELEMENTS IN THE STANDARD ROW,COLUMN FORMAT)
C   NOTE: THE A ARRAY IS ALTERED DURING EXECUTION.
C
        DIMENSION A(9,9),T(9,9),X(N),B(M)
        CC=1
C   M MUST BE <= 9, AND N<=M.  CC IS A COMPLETION CODE; IF THE
C   SUBROUTINE EXECUTES PROPERLY CC WILL BE RESET TO 0 BEFORE RETURN
        DO 40 I=1,N
          IF (I.EQ.1) GO TO 25
          DO 20 J=1,M
            S=0
            I1=I-1
            DO 10 K=1,I1
                   IF (T(K,K)  .LT. .0001) GO TO 5000
              S=S+A(J,K)*T(K,I)/T(K,K)
   10         CONTINUE
            A(J,I)=A(J,I)-S
```

```
   20     CONTINUE
   25   DO 40 K=I,N
          S=0
          DO 30 J=1,M
            S=S+A(J,I)*A(J,K)
   30       CONTINUE
          T(I,K)=S
   40     CONTINUE
        DO 60 I=1,N
          S=0
          DO 50 J=1,M
            S=S+A(J,I)*B(J)
   50       CONTINUE
          X(I)=S
   60     CONTINUE
        DO 80 I=1,N
          I1=N+1-I
          IF (I1.EQ.N) GO TO 75
            I2=I1+1
            DO 70 J=I2,N
              X(I1)=X(I1)-T(I1,J)*X(J)
   70         CONTINUE
                 IF (T(I1,I1).LT..0001) GO TO 5000
   75       X(I1)=X(I1)/T(I1,I1)
   80       CONTINUE
      CC=0
      RETURN
 5000  CC=-1
C   A COMPLETION CODE OF -1 INDICATES THAT THE SUBROUTINE
C   TRIED TO DIVIDE BY 0.
      RETURN
      END
C
C ******************************************************************
C ******************************************************************
C
        INTEGER FUNCTION IBILIN*2(XX,YY)
C
C   THIS FUNCTION RECEIVES 2 REAL COORDINATES (PRODUCED BY THE TRANS-
C   FORMATION IN THE CALLING ROUTINE) WHICH ARE COORDINATES RELATIVE
C   TO THE OLD IMAGE.   SINCE THE COORDINATES ARE REAL VALUED,
C   THE POSITION WILL NOT BE ON A PARTICULAR PIXEL, BUT RATHER AMONG
C   4.  THIS FUNCTION RETURNS A BILINEAR INTERPOLATION  FOR THE 4
C   SURROUNDING POINTS.
C
        PARAMETER (ISIZE=128,JSIZE=128)
        COMMON /IMAGE/ IMAGE(ISIZE,JSIZE)
        INTEGER*2 IMAGE
        REAL H,V,HTEMP1,HTEMP2,VTEMP
C
        MX1=MAX(0,INT(XX))
        MX2=MIN(ISIZE,INT(XX)+1)
        MY1=MAX(0,INT(YY))
        MY2=MIN(JSIZE,INT(YY)+1)
```

```
      H=XX-MX1
      V=YY-MY1
C
      HTEMP1=H*IMAGE(MX2,MY1)+(1.0-H)*IMAGE(MX1,MY1)
      HTEMP2=H*IMAGE(MX2,MY2)+(1.0-H)*IMAGE(MX1,MY2)
      VTEMP =V*HTEMP2+(1.0-V)*HTEMP1
      IBILIN=ININT(VTEMP)
      RETURN
      END
C ********************************************************************
C ********************************************************************
```

Appendix C

Adaptive Pattern Matching using Control Theory on Lie Groups

by

Thomas G. Newman and Leopold Zlobee

Proceedings of the International Symposium on the Mathematical Theory of Networks and Systems

August, 1981

ADAPTIVE PATTERN MATCHING USING CONTROL THEORY ON LIE GROUPS*

Thomas G. Newman and Leopold Zlobec
Texas Tech University
Lubbock, Texas

Abstract

A method is given for matching a subpattern of a two-dimensional image against a stored prototype, where the latter is defined on a window whose position and shape is determined by the action of a Lie group of transformations. The method involves the construction of a path in the control group along which the matching error decreases to a local minimum.

1. INTRODUCTION

A problem of classical interest in pattern recognition is that of determining the presence or absence of a particular subpattern or subpattern class. In the analysis of two-dimensional imagery this can take the form of detection of corners and edges or the location of a specific silhouette. More particularly, we may be interested in obtaining an exact match of a specific portion of the image to a subimage, often a prototype, which may appear in an arbitrary manner, varying in size, location and orientation. This is the problem which is herein addressed.

A related question was considered by Dirilten and Newman [3] where it was shown how two planar images could be matched under arbitrary affine transformation of the plane, if a match were at all possible. In addition to affine transformations, an allowance was also made for dilation of intensity scale such as that which results from under or over exposure of film within latitude limits. The results cited, however, are of little use in matching subpatterns, since the algorithms are highly sensitive to the background context. Nevertheless, the utility of a group theoretic approach to pattern matching was clearly demonstrated.

In the following we present a method for performing a local search for an imbedded subpattern of a two-dimensional image. The

*This research was supported by the Army Research Office, Contract DAAG29-80-C-0087 and by the Office of Naval Research, Contract N0014-76-C-1136.

method is one involving adaptive control of a retina which seeks the desired subpattern by evolving along a curve in the space of parameters in a direction which assures improvement in the goodness of fit.

2. BACKGROUND

Let G be a Lie group of transformation on an analytic manifold M. Suppose G has dimension n while M has dimension m. Let x and y denote the coordinates of elements f and g in G, respectively, in a patch containing the identity element e of G. Also, let p denote coordinates of an element u of M in some patch in M. We may then express the coordinates z of the product $h = fg$ and the coordinates q of the element $v = gu$, relative to suitable patches, by means of analytic functions

$$z = J(x,y) \qquad (2.1)$$

$$q = K(y,p) \qquad (2.2)$$

K and J are vector-valued, having values in n-dimensional space R^n or C^n and m-dimensional space R^m or C^m. Hereafter we shall assume that these underlying spaces are real. We denote the ith component of J by J_i and the jth component of K by K_j. In order to define the Lie algebra of G we first introduce real-valued maps on G by

$$P_{ij}(x) = \frac{\partial J_i}{\partial y_j}(x,y)\Big|_{y=e}, \qquad (2.3)$$

where i and j each range from 1 to n. The cross-section P_{*j}, which consists of the P_{ij} as i ranges from 1 to n, and j is fixed, may be thought of as a vector field in R^n. Such a vector field attaches to a point x the vector $P_{*j}(x)$. As such, P_{*1}, $P_{*2}, \ldots, P_{*n}$ form a basis for the tangent space at the point x [1,2]. The infinitesimal transformations of G may now be defined by

$$X_j = \sum_{i=1}^{n} P_{ij}(x)\frac{\partial}{\partial x_i}, \qquad (2.4)$$

for $j = 1,2,\ldots,n$.

The differential operators so defined are to be considered as linear operators on the space of analytic functions on G, or, more generally, on the space of differentiable functions on G. The Lie algebra of G is simply the n-dimensional vector space consisting of all linear combinations of these operators, and will be denoted by $L(G)$ [2]. The Lie algebra of G may also be defined in terms of its actions on the manifold M. Analogous to (2.3) we define

$$Q_{ij}(p) = \frac{\partial K_i}{\partial y_j}(y,p)\Big|_{y=e} \qquad (2.5)$$

for $i = 1,2,\ldots,m$ and $j = 1,2,\ldots,n$. Finally, as in (2.4) above we set

$$X_j' = \sum_{i=1}^{m} Q_{ij}\frac{\partial}{\partial p_i}. \qquad (2.6)$$

The operators X_1', X_2', ..., X_n' apply to functions defined on M and span a Lie algebra isomorphic to $L(G)$.

The following result from [4] will be used later, and is stated for reference:

Theorem 2.1. Let $f: M \to R$ be differentiable and define $F: G \times M \to R$, in terms of coordinates, by

$$F(x,p) = f(K(x,p)). \qquad (2.7)$$

Then for each $j = 1,2,\ldots,n$ we have

$$X_j F = X_j' F. \qquad (2.8)$$

Let us consider a curve $t \to g(t)$ in G satisfying $g(0) = e$. In terms of a coordinate patch at e, $g(t)$ may be described by a curve $x(t)$ in R^n satisfying $x(0) = 0$. We shall consider the case in which $x(t)$ is given as the solution of an evolution equation of the form

$$\dot{x}(t) = \sum_{i=1}^{n} \lambda_i(t) P_{*i}(x(t)), \; x(0) = 0, \qquad (2.9)$$

where $P_{*1}, \ldots P_{*n}$ are cross-sections of the array of functions given by (2.3), and $\lambda_1(t), \ldots, \lambda_n(t)$ are suitable control functions.

Now let p denote the coordinates of a point u in some coordinate patch. For a differentiable map $f: M \to R$ we may define $H: R \times M \to R$ by setting

$$H(t,p) = f(g(t)u). \qquad (2.10)$$

We recognize that $H(t,p) = F(x(t),p)$ where F is the extension of f to $G \times M$ as in Theorem 2.1 above. From the point of view of application, if we regard $f: M \to R$ as an image, then $H(t,p)$ represents the moving image obtained by translation due to the curve $g(t)$. Also from [4], we have

Theorem 2.2. In the context above,

$$\frac{\partial H}{\partial t} = \sum_{i=1}^{n} \lambda_i(t) X_i' H. \qquad (2.11)$$

3. THE CONTROL MODEL

By an *image* we mean a map $f: M \to R$, where the value $f(p)$ at a point $p \in M$ represents the gray value at the picture element at p. In practice, values are observed on a subset $W \subset M$, which we regard as a window which may be translated by the action of G on M. Thus, upon translation by an element $x \in G$, the value observed at $p \in W$ is given by $F(x,p) = f(X(x,p))$, as in (2.7) above.

We consider a given prototype sub-image V defined on the window W, $V: W \to R$. The problem then is to determine $x \in G$ such that $F(x,p) = V(p)$ for all $p \in W$, or determine that no such x exists. As a matter of practice, we seek $x \in G$ which minimizes the objective function

$$\Psi(x) = \frac{1}{2}\int_W (F(x,p) - V(p))^2 dp, \qquad (3.1)$$

where dp represents a volume element and the integral is over the window W, which is assumed to be of bounded volume.

In general, for any two functions f_1, f_2: $W \to M$ we define

$$\langle f_1, f_2 \rangle = \int_W f_1 f_2 dp \quad \text{and}$$

$$\|f_1\| = \langle f_1, f_1 \rangle^{1/2}.$$

Thus, $\Psi(x) = \|F - V\|^2/2$, where x is regarded as a parameter.

The following is a well-known property of the Lie group G [2]:

Lemma 1. In order that the differential $d\Psi(x) = 0$ at a point $x \in G$, it is necessary and sufficient that each $X_i\Psi(x) = 0$ where $X_1, X_2, \ldots, X_n$ are the generators of $L(G)$ given by (2.4).

By direct calculation, we obtain $X_i\Psi(x) = \int_W (F(x,p) - V(p)) X_i F(x,p) dp$. In practice, this expression is difficult to compute numerically, due to the presence of the term $X_i F$, which cannot be computed directly from observed data. However, by Theorem (2.1) we have $X_i F = X_i' F$, and the latter can be calculated from a *single value of x*.

Suppose now that a curve in G is given by coordinates $x(t)$ obtained as a solution of Equation (2.9). We seek to find $\lambda(t) = (\lambda_1(t), \ldots, \lambda_n(t))$ so that $\psi(t) = \Psi(x(t))$ decreases to a minimum value. Defining $H(t,p) = F(x(t),p)$ we obtain,

$$\dot\psi(t) = \int_W (H(t,p) - V(p)) \frac{\partial H}{\partial t}(t,p) dp \qquad (3.2)$$

which, by application of Theorem (2.2), becomes

$$\dot\psi(t) = \sum_{i=1}^{n} \lambda_i(t) \int_W (H(t,p) - V(p)) X_i' H(t,p) dp$$
$$= \sum_{i=1}^{n} \lambda_i(t) \langle H - V, X_i' H \rangle \qquad (3.3)$$

Upon observing that $\langle H - V, X_i' H \rangle = \langle F - V, X_i F \rangle = X_i \Psi$ at $x = x(t)$, we deduce:

Theorem 3.1. If $\lambda_i(t)$ is chosen so that $\operatorname{sgn}\lambda_i(t) = -\operatorname{sgn}\langle H - V, X_i' H \rangle$, we have $\dot\psi(t) \le 0$ for all t, with equality at $t = t_0$ if and only if $d\Psi = 0$ at $x = x(t_0)$.

Among the class of bounded controls, $|\lambda_i(t)| \le 1$, we see that the rate of decrease of $\psi(t)$ is maximized by the choice

$$\lambda_i(t) = -\operatorname{sgn}\langle H - V, X_i' H \rangle, \qquad (3.4)$$

for $i = 1,2,\ldots,n$. Of course, other strategies can be formulated, including steepest descent, and some methods using unbounded controls. By proceeding along trajectories defined by the solution of (2.9) with $\lambda(t)$ given by (3.4), we approach a critical point of Φ (i.e. $d = 0$). Since maxima and saddle points are unstable under perturbation, in practice this extreme point will always be a minimum.

4. SIMULATION RESULTS

The results discussed in the previous section have been implemented by a discrete algorithm and tested on simulated data [5]. A digitized two-dimensional image was first generated in the form of a large two-dimensional array, and the prototype was generated in a 20×20 window array.

The image space was assumed to be subject to translation, magnification and rotation, giving rise to a four parameter Lie group of transformations in the plane, R^2.

A number of cases were considered, including some *involving multiple (false) targets* and others in which the prototype was absent from the image being searched. In some cases the image was contaminated by 5% random noise. *In all cases the search* was started with overlap between the prototype target and the image target.

The differential equation (2.9) was solved by means of a Runge-Kutta fourth order method, with a dynamic step size, which was increased as necessary to accelerate convergence and decreased as necessary to maintain stability. Integration was replaced by summation, although we conjecture that convergence could have been accelerated by the use of a trapezoid rule.

Generally, search times ranged from 30 to 50 steps, with the longer search times prevailing for the more difficult cases.

In all cases, the final results were quite reasonable, even in those cases where the prototype was absent. In the latter cases, the search terminated with a "best" match, with a commensurately large final error.

As an example, Figure 1 shows that starting position for a noisy image containing two objects. The prototype is indicated by the central silhouette, while the true target is shifted upward, slightly to the right and is reduced in size. A false target overlaps the lower right corner of the prototype.

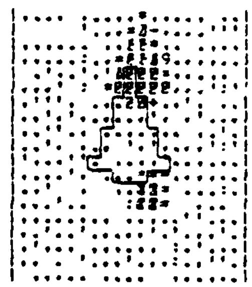


Fig. 1. Initial Window Position.

The termination conditions are shown in Figure 2, where the true target was located after 49 steps. All parameters were correct with the exception of magnification, which was about 5% too large. Smaller values of magnification, however, increase the error due to the presence of the false object, which is barely touching the bottom edge of the window in Figure 2.

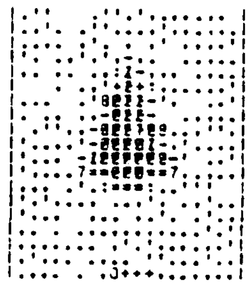


Fig. 2. Terminal Window Position.

BIBLIOGRAPHY

[1] Auslander, L., Differential Geometry, Harper and Row, New York, 1967.

[2] Cohn, P. M., Lie Groups, Cambridge University Press, London, 1957.

[3] Dirilten, H. and T. G. Newman, Pattern Matching under Affine Transformations, IEEE Trans. Comp., Vol. C-24, No. 12, 1975, pp. 1191-1201.

[4] Newman, T. G. and D. A. Demus, Lie Theoretic Methods in Video Tracking, Proceedings of the MICOM Workshop on Imaging Trackers and Autonomous Acquisition Applications, Redstone Arsenal, Nov. 1979.

[5] Zlobec, L., Pattern Matching by Means of Adaptive Control, Masters Report, Texas Tech University, 1980.

DATE
ILMED

- 8